```
manhole_cost            = zero
bur_structure           = zero
ugd_structure           = zero
aer_structure           = zero


{ Calculate soil texture impact }

soil_texture_indicator = 0

for i = 1 to NumTexTypes

    if SurfText[i].texture = soil_texture then
        soil_texture_indicator = SurfText[i].impact
    end if

next
{- should sort the SurfTest array, use binary search

{ Calculate structure distribution }

if feeder_indicator = 0 then            { this is for distribution plant }

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

else                                    { this is for feeder plant }

    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

end if
{- should set a density_index variable for use in all subsequent calculations, density does not change

{ Get sharing information }

for i = 1 to NumDensZones
    if density >= Sharing[i].density then
        ugd_share = Sharing[i].Ugd_share
        bur_share = Sharing[i].Bur_share
        aer_share = Sharing[i].Aer_share
    end if
next


{Calculate water table effect }
```



```
critical_depth = copper_placement_depth

if (copper_indicator = 1) and (fiber_indicator = 1) then
    critical_depth = max(copper_placement_depth, fiber_placement_depth) {global.pas}

if (copper_indicator = 1) and (fiber_indicator = 0) then
    critical_depth = copper_placement_depth

if (copper_indicator = 0) and (fiber_indicator = 1) then
    critical_depth = fiber_placement_depth

free_pct = one - pct_ugd - pct_bur - pct_aer
if free_pct < zero then free_pct = zero


{ Interact water table and rock hardness }

if (depth_to_bedrock < critical_depth) and (hardness = 'HARD') then
  {use hard rock values}

    for = 1 to NumDensZones
        if density >= HardRockStruc[i].density then
            if feeder_indicator = 1 then
                ugd_structure = ugd_share * HardRockStruc[i].FeedUgd
                bur_structure = bur_share * HardRockStruc[i].FeedBur
                aer_structure = aer_share * HardRockStruc[i].FeedAer
            else
                ugd_structure = ugd_share * HardRockStruc[i].DistUgd
                bur_structure = bur_share * HardRockStruc[i].DistBur
                aer_structure = aer_share * HardRockStruc[i].DistAer
            end if
        end if
    next

    //looks like feed_copper_cable_capacity is like maximum copper feeder size
    if feeder_indicator = 1 then
        NumberOfDucts = round(copper_lines / feed_copper_cable_capacity + half)
                      + round(fiber_lines / fiber_cable_capacity + half) + 1
    else
        NumberOfDucts = round(copper_lines / dist_copper_cable_capacity + half ) + 1
    end if

    if NumberOfDucts < 2 then NumberOfDucts = 2


    for i = 1 to NumDensZones
       if density >= ManholeSpac[i].density then
           ManholeSpacing = ManholeSpac[i].ManholeSpacing
       end if
    next

{attempt to find the manhole with the correct number of ducts}
    for i = 1 to NumManholeSizes - 1
        if NumberOfDucts >= ManholeCost[i].DuctCap then
            manhole_cost = ManholeCost[i].HardCost / ManholeSpacing
            { manhole cost per foot for underground}
        end if
```

```
        next

    (add any extra cost if necessary)
        if NumberOfDucts > ManholeCost[NumManholeSizes-1].DuctCap then
           manhole_cost = manhole_cost + ManholeCost[NumManholeSizes].HardCost
                        *(NumberOfDucts - ManholeCost[NumManholeSizes-1]^.DuctCap)
        end if

   else if (depth_to_bedrock >= critical_depth) and (soil_texture_indicator = 1) then
        { use normal values }
```

V - looks like soil_texture_indicator is not used properly, above test should result in 'soft rock' conditions

```
            for i = 1 to NumDensZones
                if density >= NormalStruc[i].density then
                    if feeder_indicator = 1 then
                        ugd_structure = ugd_share * NormalStruc[i].FeedUgd
                        bur_structure = bur_share * NormalStruc[i].FeedBur
                        aer_structure = aer_share * NormalStruc[i].FeedAer
                    end if

                else
                    ugd_structure = ugd_share * NormalStruc[i].DistUgd
                    bur_structure = bur_share * NormalStruc[i].DistBur
                    aer_structure = aer_share * NormalStruc[i].DistAer
                end if
            next

            if feeder_indicator = 1 then
                NumberOfDucts = round(copper_lines / feed_copper_cable_capacity + half )
                              + round(fiber_lines / fiber_cable_capacity + half) + 1
            else
                NumberOfDucts = round( copper_lines/dist_copper_cable_capacity + half ) + 1
            end if


            if NumberOfDucts < 2 then NumberOfDucts = 2

            for i = 1 to NumDensZones
                if density >= ManholeSpac[i].density then
                    ManholeSpacing = ManholeSpac[i].ManholeSpacing
                end if
            next

            for i = 1 to NumManholeSizes - 1
                if NumberOfDucts >= ManholeCost[i].DuctCap then
                    manhole_cost = ManholeCost[i].NormalCost / ManholeSpacing
                    { manhole cost per foot for underground}
                end if
            next

            if NumberOfDucts > ManholeCost[NumManholeSizes-1].DuctCap then
                manhole_cost = manhole_cost + ManholeCost[NumManholeSizes].NormalCost
                             * (NumberOfDucts - ManholeCost[NumManholeSizes-1].DuctCap)
            end if

   else  { use soft rock values }
```



```
   *W - see note above
        for i = 1 to NumDensZones
            if density >= (SoftRockStruc[i].density) then
                if feeder_indicator = 1 then
                    ugd_structure = ugd_share * SoftRockStruc[i].FeedUgd
                    bur_structure = bur_share * SoftRockStruc[i].FeedBur
                    aer_structure = aer_share * SoftRockStruc[i].FeedAer
                end if
            else
                ugd_structure = ugd_share * SoftRockStruc[i].DistUgd
                bur_structure = bur_share * SoftRockStruc[i].DistBur
                aer_structure = aer_share * SoftRockStruc[i].DistAer
            end if
        next

        if feeder_indicator = 1 then
            NumberOfDucts = round(copper_lines / feed_copper_cable_capacity + half)
                          + round(fiber_lines / fiber_cable_capacity + half) + 1
        else
            NumberOfDucts = round(copper_lines / dist_copper_cable_capacity + half) + 1

        if NumberOfDucts < 2 then NumberOfDucts = 2


        for i = 1 to NumDensZones
            if density >= ManholeSpac[i].density then
                ManholeSpacing = ManholeSpac[i].ManholeSpacing
            end if
        next


        for i = 1 to NumManholeSizes - 1
            if NumberOfDucts >= ManholeCost[i].DuctCap then
                manhole_cost = ManholeCost[i].SoftCost / ManholeSpacing
                { manhole cost per foot for underground}
            end if
        next


        if NumberOfDucts > ManholeCost[NumManholeSizes-1].DuctCap then
            manhole_cost = manhole_cost + manholeCost[NumManholeSizes].SoftCost
                         * (NumberOfDucts - ManholeCost[NumManholeSizes-1].DuctCap)
        end if
    end if
    //adjust manhole cost for sharing
    manhole_cost  = ugd_share * manhole_cost * 1000  { results in dollars per kilofoot }
    ugd_structure = ugd_structure * 1000
    bur_structure = bur_structure * 1000
    aer_structure = aer_structure * 1000

    if (MinSlope < MinSlopeTrigger) and (MaxSlope > MaxSlopeTrigger) then
        ugd_structure = ugd_structure * CombSlopeFactor
        bur_structure = bur_structure * CombSlopeFactor
        aer_structure = aer_structure * CombSlopeFactor
        manhole_cost  = manhole_cost * CombSlopeFactor
```

```
  se if (MinSlope < MinSlopeTrigger) then
    ugd_structure = ugd_structure * MinSlopeFactor
    bur_structure = bur_structure * MinSlopeFactor
    aer_structure = aer_structure * MinSlopeFactor
    manhole_cost  = manhole_cost * MinSlopeFactor

 lse if (MaxSlope > MaxSlopeTrigger) then
    ugd_structure = ugd_structure * MaxSlopeFactor
    bur_structure = bur_structure * MaxSlopeFactor
    aer_structure = aer_structure * MaxSlopeFactor
    manhole_cost  = manhole_cost * MaxSlopeFactor
nd if

//adjust percentages so they balance to 1.0, throwing any difference into plant type
with largest dollar amount
if (so_ugd_struc * ugd_structure + so_manhole * manhole_cost)
  <= min((so_bur_struc * bur_structure)                          {global.pas}
        ,(so_aer_struc * aer_structure)) then

    pct_ugd = pct_ugd + free_pct

else if (so_bur_struc * bur_structure)
  <= min((so_ugd_struc * ugd_structure + so_manhole * manhole_cost)  {global.pas}
        ,(so_aer_struc * aer_structure)) then

    pct_bur = pct_bur + free_pct

else if (so_aer_struc * aer_structure)
  <= min((so_bur_struc * bur_structure)                          {global.pas}
        ,(so_ugd_struc * ugd_structure + so_manhole * manhole_cost)) then

    pct_aer = pct_aer + free_pct

end if

ugd_structure = pct_ugd * ugd_structure
aer_structure = pct_aer * aer_structure
bur_structure = pct_bur * bur_structure
manhole_cost  = pct_ugd * manhole_cost

//this test should wrap this function, exit if there are no lines
if (copper_lines + fiber_lines) >= half then
    structure_cost_fn = ugd_structure + bur_structure + aer_structure + manhole_cost
else
    ugd_structure = zero
    bur_structure = zero
    aer_structure = zero
    manhole_cost  = zero
    structure_cost_fn = zero
end if
```



```
cable.pas

the two functions used outside of this module are feed_cable_cost and dist_cable_cost

function feed_cable_cost
    passed variables:
    lines
    density
    technology
    *ugd_copper
    *bur_copper
    *aer_copper
    *ugd_fiber
    *bur_fiber
    *aer_fiber
    pct_ugd
    pct_bur
    pct_aer


    local variables:
    i
    temp1
    temp2
    ugd2
    bur2
    aer2

    ugd_copper = zero
    bur_copper = zero
    aer_copper = zero
    ugd_fiber  = zero
    bur_fiber  = zero
    aer_fiber  = zero
    ugd2       = zero
    bur2       = zero
    aer2       = zero
    temp1      = zero
    temp2      = zero

    //lines must be total lines
    if lines < half then
        feed_cable_cost = zero

    else
        if (technology = copper26) or (technology = copper24) or (technology = t_1) then
            if lines <= feed_copper_cable_capacity then
                for i = 1 to NumFeedCableSizes
*W - looks like cables are undersized here, test should be reversed, need to use break on this loop
                    if lines >= CopDistCost[i].CableSize then
                        { costs are input per foot; we're working with kf }
                        ugd_copper = pct_ugd * CopFeedCost[i].CostUgd * 1000
                        bur_copper = pct_bur * CopFeedCost[i].CostBur * 1000
                        aer_copper = pct_aer * CopFeedCost[i].CostAer * 1000
                        temp1 = ugd_copper + bur_copper + aer_copper
                    end if
                next
```

```
        else
            for i = 1 to NumFeedCableSizes
                if (round(feed_copper_cable_capacity)) >= CopFeedCost[i].Size then
                    //this is integer division,calculating number of max size cables
                    temp1 = (round(lines) div round(feed_copper_cable_capacity))
                    ugd_copper = temp1 * pct_ugd * CopFeedCost[i].CostUgd * 1000
                    bur_copper = temp1 * pct_bur * CopFeedCost[i].CostBur * 1000
                    aer_copper = temp1 * pct_aer * CopFeedCost[i].CostAer * 1000
                    temp1 = ugd_copper + bur_copper + aer_copper
                end if
                //calculate residual cable size
                if (round(lines) mod round(feed_copper_cable_capacity))
                  >= CopFeedCost[i]^.Size then
                    ugd2 = pct_ugd * CopFeedCost[i].CostUgd * 1000
                    bur2 = pct_bur * CopFeedCost[i].CostBur * 1000
                    aer2 = pct_aer * CopFeedCost[i].CostAer * 1000
                    temp2 = ugd2 + bur2 + aer2
                end if
            next
        end if

        temp1 = temp1 + temp2
        ugd_copper = ugd_copper + ugd2
        bur_copper = bur_copper + bur2
        aer_copper = aer_copper + aer2
```

Gauge copper is assumed to be a constant multiplier of 16 gauge

```
        if technology = copper24 then
            feed_cable_cost = temp1 * multiplier_24
            ugd_copper = ugd_copper * multiplier_24
            bur_copper = bur_copper * multiplier_24
            aer_copper = aer_copper * multiplier_24
        else
            feed_cable_cost = temp1
        end if

    else { technology is fiber }

        if lines <= fiber_cable_capacity then
```

see cable sizing note above

```
            for i = 1 to NumFiberCableSizes
                if lines >= FiberFeedCost[i].size then
                    ugd_fiber = pct_ugd * FiberFeedCost[i].CostUgd * 1000
                    bur_fiber = pct_bur * FiberFeedCost[i].CostBur * 1000
                    aer_fiber = pct_aer * FiberFeedCost[i].CostAer * 1000
                    temp1 = ugd_fiber + bur_fiber + aer_fiber
                end if
            next
        else
            for i = 1 to NumFiberCableSizes
```

V - looks like this should be fiber_cable_capacity

```
                if (round(feed_copper_cable_capacity)) >= FiberFeedCost[i].Size then
```

V - looks like this should be fiber_cable_capacity

```
                    temp1 = (round(lines) div round(feed_copper_cable_capacity))
                    ugd_fiber = temp1 * pct_ugd * FiberFeedCost[i].CostUgd * 1000
                    bur_fiber = temp1 * pct_bur * FiberFeedCost[i].CostBur * 1000
```



```
                    aer_fiber = temp1 * pct_aer * FiberFeedCost[i].CostAer * 1000
                    temp1 = ugd_fiber + bur_fiber + aer_fiber
                end if
```

*W - looks like this should be fiber_cable_capacity

```
                if (round(lines) mod round(feed_copper_cable_capacity))
                  >= FiberFeedCost[i].Size then
                    ugd2 = pct_ugd * FiberFeedCost[i].CostUgd * 1000
                    bur2 = pct_bur * FiberFeedCost[i].CostBur * 1000
                    aer2 = pct_aer * FiberFeedCost[i].CostAer * 1000
                    temp2 = ugd2 + bur2 + aer2
                end if
            next
        end if

        temp1 = temp1 + temp2
        ugd_fiber = ugd_fiber + ugd2
        bur_fiber = bur_fiber + bur2
        aer_fiber = aer_fiber + aer2

        feed_cable_cost = temp1
    end if

end if

function dist_cable_cost
    passed variables:
    lines
    density
    gauge
    *ugd_copper
    *bur_copper
    *aer_copper
    pct_ugd
    pct_bur
    pct_aer


    local variables:
    i
    temp1
    temp2
    ugd2
    bur2
    aer2

    ugd_copper = zero
    bur_copper = zero
    aer_copper = zero
    temp1      = zero
    temp2      = zero
    ugd2       = zero
    bur2       = zero
    aer2       = zero

    if lines <= half then
        dist_cable_cost = zero
```

```
  else
      temp1 = zero
      temp2 = zero

      if lines <= dist_copper_cable_capacity then
          for i = 1 to NumCableSizes
V - looks like cables are undersized here, test should be reversed, need to use break on this loop
              if lines <= CopDistCost[i].CableSize then
                  ugd_copper = pct_ugd * CopDistCost[i].CostUgd * 1000
                  bur_copper = pct_bur * CopDistCost[i].CostBur * 1000
                  aer_copper = pct_aer * CopDistCost[i].CostAer * 1000
                  temp1 = ugd_copper + bur_copper + aer_copper
              end if
          next
      else
          for i = 1 to NumCableSizes
              if (round(dist_copper_cable_capacity) <= CopDistCost[i].CableSize) then
                  temp1 = (round(lines) div round(dist_copper_cable_capacity))
                  ugd_copper = temp1 * pct_ugd * CopDistCost[i].CostUgd * 1000
                  bur_copper = temp1 * pct_bur * CopDistCost[i].CostBur * 1000
                  aer_copper = temp1 * pct_aer * CopDistCost[i].CostAer * 1000
                  temp1 = ugd_copper + bur_copper + aer_copper
              end if

              if (round(lines) mod round(dist_copper_cable_capacity))
                >= CopDistCost[i].CableSize then
                  ugd2 = pct_ugd * CopDistCost[i].CostUgd * 1000
                  bur2 = pct_bur * CopDistCost[i].CostBur * 1000
                  aer2 = pct_aer * CopDistCost[i].CostAer * 1000
                  temp2 = ugd2 + bur2 + aer2
              end if
          next
      end if


      temp1 = temp1 + temp2
      ugd_copper = ugd_copper + ugd2
      bur_copper = bur_copper + bur2
      aer_copper = aer_copper + aer2

      if gauge = g24 then
          dist_cable_cost = temp1 * multiplier_24
          ugd_copper = ugd_copper * multiplier_24
          bur_copper = bur_copper * multiplier_24
          aer_copper = aer_copper * multiplier_24
      else
          dist_cable_cost = temp1
      end if

  end if
```


primdist.pas

```
the only procedure used outside of this module is calculate_prim_distribution_cost


procedure cumulate_lines
    passed variables:
    GR
    n       { number of nodes }
    _to
    line_vector
    DistToNode
    DistToSAI
    cuts
    density
    FillFactor
    *dist_cost
    *ugd_cable
    *bur_cable
    *aer_cable
    *ugd_structure
    *bur_structure
    *aer_structure
    *ManholeCost


    local variables:
    c
    i
    nc
    k
    was_cut
    g26_lines
    g24_lines
    structure_cost
    cable_cost
    technology
    uc1
    bc1
    ac1
    uc2
    bc2
    ac2
    us
    bs
    as
    mh
    penalty
    pct_ugd
    pct_bur
    pct_aer

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

```
ugd_structure = zero
bur_structure = zero
aer_structure = zero
ManholeCost   = zero
ugd_cable     = zero
bur_cable     = zero
aer_cable     = zero
uc1           = zero
bc1           = zero
ac1           = zero
uc2           = zero
bc2           = zero
ac2           = zero


nc = cuts[1]

for i = 2 to n
    if cuts[i]>nc then
        nc = cuts[i]
    end if
next

for i = 1 to n
    was_cut[i] = false
next

for i = 1 to n
    g26_lines[i] = zero
    g24_lines[i] = zero
next

for i = 2 to n
    if DistToSAI[i] > copper_gauge_xover then
        g24_lines[i] = line_vector[i]
    else
        g26_lines[i] = line_vector[i]
    end if
next

dist_cost = zero

for c = 1 to nc
    for i = 2 to n
        if not(was_cut[i]) then
            if cuts[i]=c then
                k = _to[i]
                g26_lines[k] = g26_lines[k] + g26_lines[i]
                g24_lines[k] = g24_lines[k] + g24_lines[i]

                if g26_lines[i] + g24_lines[i] > zero then
                    structure_cost = call structure_cost_fn          (structur.pas)
                                     pass variables:
                                     g26_lines[i]+g24_lines[i]
                                     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 + call dist_cable_cost(cable.pas)
                             pass variables:        pass variables:
                             g26_lines[i]           g24_lines[i]
                             density                density
                             g26                    g24
                             *uc1                   *uc2
                             *bc1                   *bc2
                             *ac1                   *ac2
                             pct_ugd                pct_ugd
                             pct_bur                pct_bur
                             pct_aer                pct_aer

            else
                cable_cost = zero
                structure_cost = zero
                uc1 = zero
                bc1 = zero
                ac1 = zero
                uc2 = zero
                bc2 = zero
                ac2 = zero
                us  = zero
                bs  = zero
                as  = zero
                mh  = zero
            end if

            if (g26_lines[i] + g24_lines[i]) > 1.0e-6 then
                if DistToSAI[i] > max_copper_distance then
                    penalty = MaxCopperPenalty
                else
                    penalty = one
                end if
*W - this penalty calculation is repeated elsewhere
                dist_cost = dist_cost + (structure_cost + cable_cost)
                          * DistToNode[i] * penalty

                ugd_cable   = ugd_cable + (uc1 + uc2) * DistToNode[i] * penalty
```

```
                    bur_cable   = bur_cable + (bc1 + bc2) * DistToNode[i] * penalty
                    aer_cable   = aer_cable + (ac1 + ac2) * DistToNode[i] * penalty

                    ugd_structure = ugd_structure + us * DistToNode[i] * penalty
                    bur_structure = bur_structure + bs * DistToNode[i] * penalty
                    aer_structure = aer_structure + as * DistToNode[i] * penalty
                    ManholeCost   = ManholeCost   + mh * DistToNode[i] * penalty

                end if

                was_cut[i] = true
            end
        end
    next
next


ɔedure prune
 passed variables:
 n      { number of nodes }
 _to
 * cut_ord


 loval variables
 total_cuts
 cut_num
 was_cut
 i
 j
 cut_it


 total_cuts = 0

 for i = 1 to n
     cut_ord[i] = 0
 next

 cut_num = 1

 for i = 1 to n
     was_cut[i] = false
 next


 repeat
     for i = 2 to n
         if not (was_cut[i]) then
             cut_it = true
             for j = 2 to n do
                 if not (was_cut[j]) then
                     if _to[j] = i then
                         cut_it = false
                     end if
```



```
                    end if
                next

                if cut_it then
                    cut_ord[i] = cut_num
                    was_cut[i] = true
                    total_cuts = total_cuts + 1
                end if
            end if
        next

        cut_num = cut_num + 1

    until total_cuts = n - 1


function provisional_cost
    passed variables:
    [illegible] {identifier for drop terminal}
    num_terms_in_SAI {total number of drop terms in SAI}
    lines
    GR
    dist            { distance to the tree }
    dist2SAI        { distance to the SAI via tree }
    density         { average density for tree }
    fillFactor


    {Calculates a provisional distribution cost for a given customer based on allocating
     both structure and cable cost between the customer and the tree, and a cable cost
      only for the entire distance from the customer to the SAI.}

    local variables:
    cable_cost
    n2016
    n672
    n96
    n24
    gauge
    uc
    bc
    ac
    us
    bs
    as
    mh
    structure_cost
    ac_structure
    ac_cable
    penalty
    pct_ugd
    pct_bur
    pct_aer
```

```
<= num_terms then, I look only at live nodes, which are indexed 1..num_terms)

{ First, make the technology determination. }

if dist2SAI > copper_gauge_xover then
    gauge = g24
else
    gauge = g26
end if


if dist2SAI > max_copper_distance then
    penalty = MaxCopperPenalty
else
    penalty = one
end if


{ Now calculate structure and cable costs.}


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


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

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



```
                 pct_aer

    ac_cable = ac_ugd_cop * uc + ac_bur_cop * bc + ac_aer_cop * ac


    if lines > 0 then
        provisional_cost = (dist * ac_structure + dist2SAI * ac_cable) * penalty
    else
        provisional_cost = zero
    end if
    else
        provisional_cost = dist/1000

    end if


procedure prim_tree
    passed variables:

    GR
    n                       { number of nodes, including SAI }
    line_vector
    dmtx                    { distance matrix}
    density                 { density for SA }
    FillFactor
    *_from                  { list of lots, with #1 = SAI }
    *_to                    { lot that each node points to }
    *dist2node              { distance from each lot to next node }
    *dist2SAI               { distance to switch from each lot }

    local constants:

    dlarge = 999999999.9

    local variables:

    i
    j
    k
    l
    a
    b
    c
    dl
    d2s
    min
    idx
    dist
    dist2
    cost
    technology

    for i = 1 to n do
        a[i] = true
        b[i] = 0
```

t.pas

```
    c[i] = dlarge
    d1[i] = zero
 xt

 1] = zero
 s[1] = zero

or i = 2 to n
    d2s[i] = dmtx[i][1]
 ext

 = 1

or i = 2 to n
    min = dlarge
    for k = 2 to n
        if (k <> j) then
            if s[k] then
                dist = dmtx[j][k]
                cost = call provisional_cost
                        pass variables:
                        [illegible] = k
                        num_terms = num_terms
                        lines     = line_vector[k]
                        GR        = GR
                        dist      = dist
                        dist2SAI  = dist + d2s[j]
                        density   = density
                        FillFactor = FillFactor


                if cost < c[k] then
                    c[k]  = cost
                    d1[k] = dist
                    b[k]  = j
                end  if

                if min > c[k] then
                    min = c[k]
                    l = k
                    dist2 = d1[k] + d2s[b[k]]
                end if
            end if  { if s[k] }
        end if
    next k

    j = 1
    s[j] = false
    d2s[1] = dist2

 next i

for i = 2 to n do
    _from[i] = i
    _to[i] = b[i]
```

(primdist.pas)

primdist.pas

```
        dist2node[i] = d1[i]
        dist2SAI[i] = d2s[i]
    next

    _from[1] = 1
    _to[1] = 1
    dist2node[1] = zero
    dist2SAI[1] = zero


procedure get_lines
    passed variables:
    GR
    density
    row
    col
    NS_lots
    EW_lots
    lines
    *n
    *line_vector
    *x
    *y
    *drop_terminal_cost
    *HG_nid_cost
    *drop_cost
    *drop_feet

    local variables:
    i
    j
    factor
    lines_per_lot
    total_lots
    drop_length
    pct_ugd
    pct_bur
    pct_aer
    tmp
    us
    bs
    as
    mh


    drop_terminal_cost = zero
    total_lots = EW_lots * NS_lots
    lines_per_lot = lines / total_lots

    i = 1
    loop while i <= EW_lots

        factor = one
        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 (i = EW_lots) or (j = NS_lots) then
        factor = 2
    else
        factor = 4
    end if

    if (i = EW_lots) and (j = NS_lots) then
        factor = one
    end if

    n = n+1

    line_vector[n] = factor * lines_per_lot

    x[n] = GR.LowerLeftX + (col - 1) * GR.MicroGridEW + i * (one / EW_lots)
    y[n] = GR.LowerLeftY + (row - 1) * GR.MicroGridNS + j * (one / NS_lots)

    tmp = call structure_cost_fn                                  (structur.pas)
          pass variables:
          line_vector[n]
          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


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

    j = j + 2
end loop  (j <= NS_lots)

    i = i+2
end  { while i }
```



```
    { 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) * .5 * (1 / NS_lots) * GR.MicroGridNS * DistRoadFactor

    if drop_length > max_drop_length then
        drop_length = max_drop_length
    End if

    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_prim_distribution_cost
    passed variables:
    GR
    num_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
    *MaximumDistance
    *ugd_cable
    *bur_cable
    *aer_cable
    *ugd_structure
    *bur_structure
    *aer_structure
    *ManholeCost

    local variables:
    i
    j
    n
    k
    midx
    midy
    lots
    EW_lots
    NS_lots
```

```
area
cable_cost
structure_cost
total_lines
SAI_LinkDistance
tl_lines
gauge
penalty
uc
bc
ac
us
bs
as
mh
dmtx
x
y
line_vector
_from
_to
cut_ord
DistToNode
DistToSAI
route_distance
tcost
grid_dropterm_cost
grid_drop_cost
grid_nid_cost
grid_drop_feet
dist_cost
test


prim_distribution_cost - zero
prim_drop_cost         - zero
prim_nid_cost          - zero
prim_term_cost         - zero
prim_drop_feet         - zero
ugd_cable              - zero
bur_cable              - zero
aer_cable              - zero
ugd_structure          - zero
bur_structure          - zero
aer_structure          - zero
HanholeCost            - zero
MaximumDistance        - zero
prim_line_feet         - zero


for k = 1 to num_SAIs
    { First, we need to set up the distance matrix to be used by PrimTree. }

    x[1] = SAIX[k]
    y[1] = SAIY[k]
    line_vector[1] = zero
    n = 1
```



```
for i = 1 to GR.nrow
    for j = 1 to GR.ncol
        if (flag[i,j] = k) and (lines[i,j] > zero) then
            lots = round(GR.households[i,j] * takerate)
                 + round(GR.buslines[i,j] / lines_per_bus)

            call lot_divide                                    (lotdiv.pas)
            pass variables:
            lots
            *NS_lots
            *EW_lots

            call get_lines                                     (primdist.pas)
            pass variables:
            GR
            density
            i
            j
            round(NS_lots)
            round(EW_lots)
            lines[i,j]
            *n
            *line_vector
            *x
            *y
            *grid_dropterm_cost
            *grid_nid_cost
            *grid_drop_cost
            *grid_drop_feet

            prim_drop_cost - prim_drop_cost + grid_drop_cost
            prim_nid_cost  - prim_nid_cost  + grid_nid_cost
            prim_term_cost - prim_term_cost + grid_dropterm_cost
            prim_drop_feet - prim_drop_feet + grid_drop_feet
        end if
    next j
next i


num_terms = n - 1 {number of drop terminals, not including SAI}


{Add Steiner junction nodes at each lattice point along axes through SAI location}

for i = 1 to GR.ncol
    x[n+i] = GR.LowerLeftX + (i - 1) * GR.MicroGridEW
    y[n+i] = y[1]
    line_vector[n+i] = zero
next

for i = 1 to GR.nrow
    x[n+GR.ncol+i] = x[1]
    y[n+GR.ncol+i] = GR.LowerLeftY + (i - 1) * GR.MicroGridNS
    line_vector[n+GR.ncol+i] = zero
next
```

```
n = n + GR.ncol * GR.nrow
num_terms = num_terms + y[ ( we want the number of live nodes to include the SAI )


for i = 1 to n do
    for j = 1 to i do
        dmtx[i][j] = (abs(x[i] - x[j]) + abs(y[i] - y[j])) * DistRoadFactor
        dmtx[j][i] = dmtx[i][j]
    next j
next i

( Now calculate the spanning tree. )

call prim_tree                                          (primdist.pas)
pass variables:
GR
n
line_vector
dmtx
density
FillFactor
*_from
*_to
*DistToNode
*DistToSAI


( Determine cuts by pruning branches )

call prune                                              (primdist.pas)
pass variables:
n
_to
*cut_ord

( Accumulate lines at each node, and sum total feeder cost )


call cumulate_lines                                     (primdist.pas)
pass variables:
GR
n
_to
line_vector
DistToNode
DistToSAI
cut_ord
density
FillFactor
*dist_cost
*uc
*bc
*ac
*us
*bs
*as
```



```
*mh


prim_distribution_cost = prim_distribution_cost + dist_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

for i = 1 to n
    if DistToSAI[i] > MaximumDistance then MaximumDistance = DistToSAI[i]
next

for i = 1 to n
    prim_line_feet = prim_line_feet + line_vector[i] * DistToSAI[i]
next


next k
```

```
rimfeed.pas

he three procedures available outside of this module are:
  cumulate_lines
  prune
  prim_tree


rocedure cumulate_lines
   passed variables:
   n  { number of nodes }
   _to
   DistToNode
   DistToSwitch
   cuts
   structure_cost
   density
   FillFactor
   *feeder_cost
   *feed_splice_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

   local variables:
   c
   i
   nc
   k
   was_cut
   g26_lines
   g24_lines
   t1_lines
   fiber_lines
   fiber_cable
   cable_cost
   technology
   n2016
   n672
   n96
   n24
   uc1
```

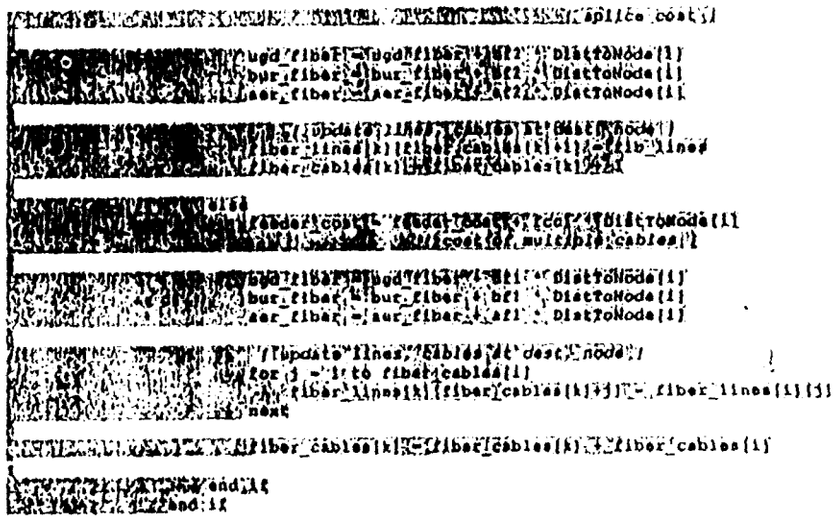


```
   bc1
   ac1
   uf1
   bf1
   af1
   uc2
   bc2
   ac2
   uf2
   bf2
   af2
   uc3
   bc3
   ac3
   uf3
   bf3
   af3
   uc4
   bc4
   ac4
   df4
   bf4
   af4
   us
   bs
   as
   mh
   fcc1
   fcc2
   fib_lines


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

   ugd_structure = zero
   bur_structure = zero
   aer_structure = zero
   ManholeCost   = zero
   ugd_cable     = zero
   bur_cable     = zero
   aer_cable     = zero
   ugd_fiber     = zero
   bur_fiber     = zero
   aer_fiber     = zero
   uc1           = zero
   bc1           = zero
   ac1           = zero
   uf1           = zero
   bf1           = zero
   af1           = zero
   uc2           = zero
   bc2           = zero
   ac2           = zero
   uf2           = zero
```

```
bf2             = zero
af2             = zero
uc3             = zero
bc3             = zero
ac3             = zero
uf3             = zero
bf3             = zero
af3             = zero
uc4             = zero
bc4             = zero
ac4             = zero
uf4             = zero
bf4             = zero
af4             = zero


{ First, make the technology determination. }

for i = 2 to n
    if i <= num_Sas + 1 then
        call calculate_feeder_technology                    (tech.pas)
        pass variables:
        DistToSwitch[i]
        i - 1
        density
        FillFactor
        *technology
        *n2016
        *n672
        *n96
        *n24
        pct_ugd
        pct_bur
        pct_aer

        nc = cuts[i] for i = 2 to n do if cuts[i]>nc then nc = cuts[i]
    end if
next

for i = 1 to n
    was_cut[i] = false
next



for i = 1 to n
    g26_lines[i] = ZERO
    g24_lines[i] = ZERO
    t1_lines[i] = ZERO
    fiber_cables[i] = 0
    for j = 1 to 100
        fiber_lines[i][j] = ZERO
    next
next

for i = 2 to n
    if i <= num_Sas + 1 then
```



```
        select case SA_array[i-1].feeder_technology
            case copper26
                g26_lines[i] = SA_array[i-1].ResLines / FillFactor
                             +(SA_array[i-1].BusLines
                              - 11 / 12 * SA_array[i-1].SwitchedDS1
                              - 11 / 12 * SA_array[i-1].SpclAccessDS1) / FillFactor

                t1_lines[i] = (SA_array[i-1].SwitchedDS1
                             +  SA_array[i-1].SpclAccessDS1) / FillFactor

            case copper24
                g24_lines[i] = SA_array[i-1].ResLines / FillFactor
                             +(SA_array[i-1].BusLines
                              - 11 / 12 * SA_array[i-1].SwitchedDS1
                              - 11 / 12 * SA_array[i-1].SpclAccessDS1) / FillFactor

                t1_lines[i] = (SA_array[i-1].SwitchedDS1
                             +  SA_array[i-1].SpclAccessDS1) / FillFactor


            case t_1
                t1_lines[i] = (SA_array[i-1].ResLines / FillFactor
                             +  SA_array[i-1].BusLines / FillFactor)
                             * t1_redundancy_factor / 12

            case fiber
                fiber_lines[i] = (SA_array[i-1].n2016 + SA_array[i-1].n672
                               +  SA_array[i-1].n96 + SA_array[i-1].n24)
                               * 4 / FiberFillFactor
        end select
    next
end if

feeder_cost = ZERO
feed_splice_cost = ZERO

for c = 1 to nc
    for i = 2 to n
        if not(was_cut[i]) then
            if cuts[i]=c then
                k = _to[i]
                g26_lines[k] = g26_lines[k] + g26_lines[i]
                g24_lines[k] = g24_lines[k] + g24_lines[i]
                t1_lines[k] = t1_lines[k] + t1_lines[i]

                {do fiber lines first}

                {we look at splicing cost where economically efficient}

                if fiber_cables[i] > 0 then
                    fcol = zero
                    uf1 = zero
                    bf1 = zero
                    af1 = zero
                    uf2 = zero
                    bf2 = zero
```



```
cable_cost = call feed_cable_cost                    (cable.pas)
             pass variables:
             g26_lines[i]
             density
             copper26
             *uc1
             *bc1
             *ac1
             *uf1
             *bf1
             *af1
             pct_ugd
             pct_bur
             pct_aer

           + call feed_cable_cost                    (cable.pas)
             pass variables:
             g24_lines[i]
             density
             copper24
             *uc2
             *bc2
             *ac2
             *uf2
             *bf2
             *af2
             pct_ugd
```

```
                    pct_bur
                    pct_aer

                  * call feed_cable_cost              (cable.pas)
                    pass variables:
                    t1_lines[i]
                    density
                    t_l
                    *uc3
                    *bc3
                    *ac3
                    *uf3
                    *bf3
                    *af3
                    pct_ugd
                    pct_bur
                    pct_aer

                  * call feed_cable_cost              (cable.pas)
                    pass variables:
                    fiber_lines[i]
                    density
                    fiber
                    *uc4
                    *bc4
                    *ac4
                    *uf4
                    *bf4
                    *af4
                    pct_ugd
                    pct_bur
                    pct_aer

            if (g26_lines[i] + g24_lines[i] + t1_lines[i]+ fiber_cables[i])
             > 1.0e-6 then
                feeder_cost = feeder_cost + (structure_cost + cable_cost)
                            * DistToNode[i]

                ugd_cable = ugd_cable + (uc1 + uc2 + uc3 + uc4) * DistToNode[i]
                bur_cable = bur_cable + (bc1 + bc2 + bc3 + bc4) * DistToNode[i]
                aer_cable = aer_cable + (ac1 + ac2 + ac3 + ac4) * DistToNode[i]
                ugd_fiber = ugd_fiber + (uf1 + uf2 + uf3 + uf4) * DistToNode[i]
                bur_fiber = bur_fiber + (bf1 + bf2 + bf3 + bf4) * DistToNode[i]
                aer_fiber = aer_fiber + (af1 + af2 + af3 + af4) * DistToNode[i]
                ugd_structure = ugd_structure + us * DistToNode[i]
                bur_structure = bur_structure + bs * DistToNode[i]
                aer_structure = aer_structure + as * DistToNode[i]
                ManholeCost   = ManholeCost   + mh * DistToNode[i]

            end if

            was_cut[i] = true
        end if
    end if
  next i
next c
```



```
procedure prune
    passed variables:
    n              { number of nodes }
    _to
    *cut_ord


    local variables:
    total_cuts
    cut_num
    was_cut
    i
    j
    cut_it


    total_cuts = 0

    for i = 1 to n
        cut_ord[i] = 0
    next

    cut_num = 1

    for i = 1 to n
        was_cut[i] = false
    next

    repeat
        for i = 2 to n
            if not (was_cut[i]) then
                cut_it = true
                for j = 2 to n
                    if not (was_cut[j]) then
                        if _to[j] = i then
                            cut_it = false
                        end if
                    end if
                next

                if cut_it then
                    cut_ord[i] = cut_num
                    was_cut[i] = true
                    total_cuts = total_cuts + 1
                end if
            end if

            cut_num = cut_num + 1

    until total_cuts = n - 1


function provisional_cost
    passed variables:
```

```
i                   { indexes the SAI }
dist                { distance to the tree }
dist2switch         { distance to the switch via tree }
density             { average density for tree }
fillfactor
ac_structure
pct_ugd
pct_bur
pct_aer


local variables:
cable_cost
n2016
n672
n96
n24
lines
technology
uc
bc
ac
uf
bf
af
sai_index

{Calculates a provisional feeder cost for a given SAI based on allocating both
structure and cable cost between the SAI and the tree, and a cable cost only for
the entire distance from the SAI to the switch. }



if i <= num_SAs then  {look only at "live" nodes, which are indexed 1..num_SAs }

    { First, make the technology determination. }

    call calculate_feeder_technology                          {tech.pas}
    pass variables:
    dist2switch
    i
    density
    fillfactor
    *technology
    *n2016
    *n672
    *n96
    *n24
    pct_ugd
    pct_bur
    pct_aer


    { Now calculate structure and cable costs, assuming that geologic factors
     throughout feeder route are the same as those for this SA. }
```



```
        select case technology

            case fiber
                lines = (n2016 + n672 + n96 + n24) * 4 / FiberFillFactor

            case t_1
                lines = SA_array[i].lines * t1_redundancy_factor / 12

            case else
                lines = SA_array[i].ResLines / FillFactor + (SA_array[i].BusLines
                      - 11 / 12 * SA_array[i].SwitchedDS1
                      - 11 / 12 * SA_array[i].SpclAccessDS1) / FillFactor
        end select

        cable_cost = call feed_cable_cost                          {cable.pas}
                     pass variables:
                     lines
                     density
                     technology
                     *uc
                     *bc
                     *ac
                     *uf
                     *bf
                     *af
                     pct_ugd
                     pct_bur
                     pct_aer


        if lines >   then
            provisional_cost = dist * ac_structure + dist2switch
                             * (ac_ugd_cop * uc + ac_bur_cop * bc + ac_aer_cop * ac
                             +  ac_ugd_fib * uf + ac_bur_fib * bf + ac_aer_fib * af)
        else
            provisional_cost = zero
        end if

    else

        provisional_cost = dist / 1000

    end if



procedure prim_tree
    passed variables:
    n               { number of nodes, including switch }
    dmtx            { distance matrix }
    density         { average density for tree }
    fillfactor
    ac_structure
    *_from          { list of SAIs, with #1 - switch }
    *_to            { SAI that each node points to }
    *dist2node      { distance from each SAI to next node }
```

```
*dist2switch    { distance to switch from each SAI }
pct_ugd
pct_bur
pct_aer

local constants:
dlarge = 999999999.9

local variables:
i
j
k
l
a
b
c
d1
d2s
min
idx
dist
dist2
cost
technology

for i = 1 to n do
    a[i] = true
    b[i] = 0
    c[i] = dlarge
    d1[i] = zero
next

c[1] = zero
d2s[1] = zero

for i = 2 to n
    d2s[i] = dmtx[i][1]
next

j = 1

for i = 2 to n
    min = dlarge
    for k = 2 to n
        if (k <> j) then
            if a[k] then
                dist = dmtx[j][k]
                cost = call provisional_cost
                       pass variables:
                       k-1
                       dist
                       dist+d2s[j]
                       density
                       FillFactor
                       ac_structure
                       pct_ugd
                       pct_bur
```



```
                       pct_aer

                    if cost < c[k] then
                        c[k]  = cost
                        d1[k] = dist
                        b[k]  = j
                    end if

                    if min > c[k] then
                        min = c[k]
                        l = k
                        dist2 = d1[k] + d2s[ b[k] ]
                    end if
                end if
            end if
        next k

        j = l
        a[j] = false
        d2s[l] = dist2

    next i

    for i = 2 to n
        _from[i] = i
        _to[i] = b[i]
        dist2node[i] = d1[i]
        dist2switch[i] = d2s[i]
    next

    _from[1] = 1
    _to[1] = 1
    dist2node[1] = zero
    dist2switch[1] = zero
```

```
rimsai.pas

he only procedure used outside of this function is get_link_cost

rocedure cumulate_lines
   passed variables:
   n              ( number of nodes )
   _to
   ds0_lines
   DistToNode
   DistToPrimary
   cuts
   density
   SA
   *link_cost
   *term_cost
   *n96
   *n24
   *ugd_cable
   *bur_cable
   *aer_cable
   *ugd_structure
   *bur_structure
   *aer_structure
   *ManholeCost

   local variables
   c
   i
   nc
   k
   was_cut
   t1_lines
   cable_cost
   technology
   structure_cost
   uc
   bc
   ac
   uf
   bf
   af
   us
   bs
   as
   mh
   tmp
   pct_ugd
   pct_bur
   pct_aer


   technology = t_1

   nc = cuts[1]
```



```
   for i = 2 to n
      if cuts[i]>nc then
         nc = cuts[i]
      end if

   for i = 1 to n
      was_cut[i] = false
   next

   for i = 1 to n
      t1_lines[i] = zero
   next

   for i = 2 to n
      t1_lines[i] = ds0_lines[i] * t1_redundancy_factor / 12
   next

   link_cost      = zero
   term_cost      = zero
   ugd_cable      = zero
   bur_cable      = zero
   aer_cable      = zero
   ugd_structure  = zero
   bur_structure  = zero
   aer_structure  = zero
   ManholeCost    = zero

   for c = 1 to nc
      for i = 2 to n
         if not(was_cut[i]) then
            if cuts[i]=c then
               k = _to[i]
               t1_lines[k] = t1_lines[k] + t1_lines[i]
               if t1_lines[i] > half then
                  tmp = call structure_cost_fn                    (structur.pas)
                        pass variables:
                        t1_lines[i]
                        0
                        density
                        SA.hardness
                        SA.DepthToBudrock
                        SA.SoilTexture
                        SA.MinSlope
                        SA.MaxSlope
                        SA.WaterTb
                        1
                        1
                        0
                        *us
                        *bs
                        *as
                        *mh
                        *pct_ugd
                        *pct_bur
                        *pct_aer

               else
```

```
    tmp = call structure_cost_fn              (structur.pas)
          pass variables:
          9999
          0
          density
          SA.hardness
          SA.DepthToBedrock
          SA.SoilTexture
          SA.MinSlope
          SA.MaxSlope
          SA.WaterTb
          1
          1
          0
          *us
          *bs
          *as
          *mh
          *pct_ugd
          *pct_bur
          *pct_aer
end if

structure_cost = tmp


if tl_lines[i] > half then
    tmp = call feed_cable_cost                (cable.pas)
          pass variables:
          tl_lines[i]
          density
          t_l
          *uc
          *bc
          *ac
          *uf
          *bf
          *af
          pct_ugd
          pct_bur
          pct_aer

else
    tmp = call feed_cable_cost                (cable.pas)
          pass variables:
          9999
          density
          t_l
          *uc
          *bc
          *ac
          *uf
          *bf
          *af
          pct_ugd
          pct_bur
          pct_aer
```



```
end if

cable_cost = tmp

if tl_lines[i] > half then
    tmp = call tl_terminal_cost_fn            (terminal.pas)
          pass variables:
          ds0_lines[i]
          *n96
          *n24
else
    tmp = ac24
    n96 = 0
    n24 = 1
end if

term_cost = term_cost + tmp

link_cost = link_cost + (structure_cost + cable_cost)
          * DistToNode[i] * DistRoadFactor

ugd_cable     = ugd_cable + uc * DistToNode[i] *DistRoadFactor
bur_cable     = bur_cable + bc * DistToNode[i] * DistRoadFactor
aer_cable     = aer_cable + ac * DistToNode[i] * DistRoadFactor
ugd_structure = ugd_structure + us * DistToNode[i] * DistRoadFactor
bur_structure = bur_structure + bs * DistToNode[i] * DistRoadFactor
aer_structure = aer_structure + as * DistToNode[i] * DistRoadFactor
ManholeCost   = ManholeCost   + mh * DistToNode[i] * DistRoadFactor

was_cut[i] = true
end if
end if
next i
next t


procedure prune
    passed variables:
    n      { number of nodes }
    _to
    *cut_ord

    local variables:
    total_cuts
    cut_num
    was_cut
    i
    j
    cut_it


    total_cuts = 0

    for i = 1 to n
        cut_ord[i] = 0
    next
```

```
    cut_num = 1

    for i = 1 to n
        was_cut[i] = false
    next

    repeat
        for i = 2 to n
            if not (was_cut[i]) then
                cut_it = true
                for j = 2 to n
                    if not(was_cut[j]) then
                        if _to[j]=i then
                            cut_it = false
                        end if
                    end if
                next j

                if cut_it then
                    cut_ord[i] = cut_num
                    was_cut[i] = true
                    total_cuts = total_cuts + 1
                end if
            end if
        next i

        cut_num = cut_num + 1

    until total_cuts = n - 1


rocedure prim_tree
    passed variables:
    n               { number of nodes, including primary SAI }
    dmtx            { upper triangle of distance matrix}
    *_from          { list of SAIs, with #1 = primary }
    *_to            { Bring forward lines from previous microgrids }
    *d1             { distance from each SAI to next node }
    *d2p            { distance to primary from each SAI }

    local constants:
    dlarge = 999999999.9

    local variables:
    i
    j
    k
    l
    a
    b
    c
    min
    dist
    dist2
    cost
```



```
    for i = 1 to n do
        a[i] = true
        b[i] = 0
        c[i] = dlarge
        d1[i] = zero
    next

    c[1] = zero
    d2p[1] = zero

    for i = 2 to n
        d2p[i] = dmtx[i][1]
    next

    j = 1

    for i = 2 to n
        min = dlarge
        for k = 2 to n
            if (k <> j) then
                if a[k] then
                    dist = dmtx[j][k]
                    cost = dist
                    if cost < c[k] then
                        c[k]  = cost
                        d1[k] = dist
                        b[k]  = j
                    end if

                    if min > c[k] then
                        min = c[k]
                        l = k
                        dist2 = d1[k] + d2p[b[k]]
                    end if
                end if
            end if
        next k

       j = l
       a[j] = false
       d2p[i] = dist2

    next i

    for i = 2 to n do
        _from[i] = i
        _to[i] = b[i]
    next

    _from[1] = 1
    _to[1] = 1


procedure get_link_cost
    passed variables:
    number_of_SAIS
```

```
SA
SAI_lines
saix
saiy
density
*link_cost
*term_cost
*link_line_feet
*nc96
*nc24
*ugd_cable
*bur_cable
*aer_cable
*ugd_structure
*bur_structure
*aer_structure
*HanholeCost

local variables:
uc
bc
ac
us
bs
as
mh
m
_from
_to
DistToNextTerm
DistToPrimary
cuts
i
j

term_cost = zero
link_cost = zero
ugd_cable = zero
bur_cable = zero
aer_cable = zero
ugd_structure = zero
bur_structure = zero
aer_structure = zero
HanholeCost   = zero
link_line_feet = zero

if number_of_SAIs > 1 then

    { First, set up distance matrix between SAIs...}
    for i = 1 to number_of_SAIs
        for j = 1 to number_of_SAIs do
            m[i][j] = abs(saix[i]-saix[j]) + abs(saiy[i]-saiy[j])
        next j
    next i

    call prim_tree                                    (primsai.pas)
    pass variables:
```



```
    n      = number_of_SAIs
    dmtx   = m
    *_from = _from
    *_to   = _to
    *d1    = DistToNextTerm
    *d2    = DistToPrimary

    call prune                                        (primsai.pas)
    pass variables:
    n        = number_of_SAIs
    _to      = _to
    *cut_ord = cuts

    call cumulate_lines                               (primsai.pas)
    pass variables:
    n              = number_of_SAIs
    _to            = _to
    ds0_lines      = SAI_lines
    DistToNode     = DistToNextTerm
    DistToPrimary  = DistToPrimary
    cuts           = cuts
    density        = density
    SA             = SA
    *link_cost     = link_cost
    *term_cost     = term_cost
    *n96           = nc96
    *n24           = nc24
    *ugd_cable     = uc
    *bur_cable     = bc
    *aer_cable     = ac
    *ugd_structure = us
    *bur_structure = bs
    *aer_structure = as
    *Hanhole_cost  = mh

    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
    HanholeCost   = HanholeCost   + mh

    for i = 1 to number_of_SAIs
        link_line_feet = link_line_feet + SAI_lines[i] * DistToPrimary[i]
    next

end if
```

```
rminal.pas

ree functions are used outside of this module:

  fiber_terminal_cost_fn
  t1_terminal_cost_fn
  drop_terminal_cost_fn


notion fiber_terminal_cost_fn
  passed variables:
  lines
  distance
  density
  *n2016
  *n672
  *n96
  *n24
  pct_ugd
  pct_bur
  pct_aer

  local variables
  cost
  mincost
  min2016
  min672
  min96
  min24
  i
  j
  k
  12016
  1672
  196
  124
  cabcost
  uc
  bc
  ac
  uf
  bf
  af

  {Calculates cost of fiber terminals for a given number of DS0 lines served,
   including number of terminals of each size, using integer search. }

  if lines > half then
      mincost = 1.0e+16
      for i = 0 to round( lines / 2016.0 + half )
          for j = 0 to round( (lines - 2016.0 * i) /672 + half )
              for k = 0 to round( (lines - 2016 *i - 672 * j) / 96 + half )
                  n2016 = i
                  n672 = j
                  n96 = k
                  n24 = round((lines - 2016 * n2016 - 672 * n672 - 96 * n96)/24 + half )
                  if n24 < 0 then n24 = 0
```



```
                  12016 = min(2016.0 * n2016, lines )                    {global.pas}
                  1672 = min( 672.0 * n672, lines - 12016 )
                  196 = min( 96.0 * n96, lines - 12016 - 1672 )
                  124 = lines - 12016 - 1672 - 196
                  cost = a2016 * n2016 + b2016 * 12016 + a672 * n672
                       + b672 * 1672 + a96 * n96 + b96 * 196 + a24 * n24 + b24 * 124

                  cabcost = call feed_cable_cost                         {cable.pas}
                            pass variables:
                            lines        =(n2016 + n672 + n96 + n24) * 4 / FiberFillFactor
                            density      = density
                            technology   = fiber
                            *ugd_copper  = uc
                            *bur_copper  = bc
                            *aer_copper  = ac
                            *ugd_fiber   = uf
                            *bur_fiber   = bf
                            *aer_fiber   = af
                            pct_ugd      = pct_ugd
                            pct_bur      = pct_bur
                            pct_aer      = pct_aer

                  cost = cost + cabcost * distance

                  if cost < mincost then
                      mincost = cost
                      min2016 = n2016
                      min672 = n672
                      min96 = n96
                      min24 = n24
                  end if

              next k
          next j
      next i

      n2016 = min2016
      n672 = min672
      n96 = min96
      n24 = min24

      cabcost = call feed_cable_cost                                     {cable.pas}
      pass variables:
      lines        = (n2016 + n672 + n96 + n24) * 4 / FiberFillFactor
      density      = density
      technology   = fiber
      *ugd_copper  = uc
      *bur_copper  = bc
      *aer_copper  = ac
      *ugd_fiber   = uf
      *bur_fiber   = bf
      *aer_fiber   = af
      pct_ugd      = pct_ugd
      pct_bur      = pct_bur
      pct_aer      = pct_aer
```

```
    fiber_terminal_cost_fn = mincost - cabcost * distance

 else
    n2016 = 0
    n672 = 0
    n96 = 0
    n24 = 0

    fiber_terminal_cost_fn = zero

 end if



otion t1_terminal_cost_fn
 passed variables:
 lines
 *nc96
 *nc24

 local variables:
 cost
 mincost
 min96
 min24
 i
 196
 124


 {Calculates cost of t-1 terminals for a given number of DS0 lines served, including
      number of terminals of each size, using integer search. }

 if lines > half then
    mincost = 1.0e+16
    for i = 0 to round(lines / 96 + half )
        nc96 = i
        nc24 = round((lines - 96 * nc96) / 24 + half )
        if nc24 < 0 then nc24 = 0

        196 = min(96 * nc96, lines)                          {global.pas}
        124 = lines - 196

        cost = ac96 * nc96 + bc96 * 196 + ac24 * nc24 + bc24 * 124

        if cost < mincost then
            mincost = cost
            min96 = nc96
            min24 = nc24
        end if
    next

    nc96 = min96
    nc24 = min24
    t1_terminal_cost_fn = mincost

 else
```



```
        nc96 = 0
        nc24 = 0
        t1_terminal_cost_fn = zero

    end if


function drop_terminal_cost_fn
    passed variables:
    lines
    density
    pct_ugd
    pct_bur
    pct_aer

    local variables:
    i
    temp

    temp = zero

    if lines < 1.0e-6 then
        drop_terminal_cost_fn = zero
    else

        temp = zero
        for i = 1 to NumDropTerminalSizes
            if lines >= DropTermCost[i].size then
                temp = pct_ugd * DropTermCost[i].CostUgd
                     + pct_bur * DropTermCost[i].CostBur
                     + pct_aer * DropTermCost[i].CostAer
            end if
        next

        drop_terminal_cost_fn = temp

    end if
```

```
ech.pas

ie only procedure used outside of this modules is calculate_feeder_technology


rocedure calculate_feeder_technology
    passed variables:
    feeder_distance
    i
    density
    FillFactor
    *technology
    *n2016
    *n672
    *n96
    *n24
    pct_ugd
    pct_bur
    pct_aer


    local variables:
    n
    tmp1
    tmp2
    tmp3
    c26
    c24
    ct1
    cf
    126
    124
    lt1
    lf
    uc
    bc
    ac
    uf
    bf
    af

    n2016 = 0
    n672 = 0
    n96 = 0
    n24 = 0
    technology = copper26

    SA_array[i].fiber_terminal_cost = zero
    SA_array[i].t1_terminal_cost = zero
    SA_array[i].interface_cost = zero

    SA_array[i].n2016 = 0
    SA_array[i].n672 = 0
    SA_array[i].n96 = 0
    SA_array[i].n24 = 0
    SA_array[i].nc96 = 0
```



```
    SA_array[i].nc24 = 0

    126 = SA_array[i].ResLines / FillFactor
        +(SA_array[i].BusLines - 11 / 12 * SA_array[i].SwitchedDS1
        - 11 / 12 * SA_array[i].SpclAccessDS1) / FillFactor

    124 = 126

    lt1 = (SA_array[i].ResLines / FillFactor + SA_array[i].BusLines / FillFactor)
       * t1_redundancy_factor / 12

    tmp1 = call fiber_terminal_cost_fn                                  (terminal.pas)
           pass variables:
           lines    = SA_array[i].lines / FillFactor
           distance = feeder_distance
           density  = SA_array[i].density
           *n2016   = n2016
           *n672    = n672
           *n96     = n96
           *n24     = n24
           pct_ugd  = pct_ugd
           pct_bur  = pct_bur
           pct_aer  = pct_aer

    tmp1 = tmp1 * so_fib_term

    lf = (n2016 + n672 + n96 + n24) * 4 / FiberFillFactor

    { Calculate provisional terminal costs.  Note that the terminal cost fns use DS0
    equivalent lines, so we need the fill factor, but not DS1 calculations.}

    tmp2 = call t1_terminal_cost_fn                                     (terminal.pas)
           pass variables:
           lines = SA_array[i].lines / FillFactor
           *nc96 = n96
           *nc24 = n24

    tmp2 = tmp2 * so_t1_term

    tmp3 = zero

    for n = 1 to NumICBoxLines
        if 126 >= IntfcCost[n].NumLines then
            tmp3 = IntfcCost[n].cost
        end if
    next

    tmp3 = tmp3*so_fdi


    { We will choose feeder technology by least-cost under the assumption that each
     SA sends feeder directly  to the switch without sharing cable. }

    c26 = call feed_cable_cost                                          (cable.pas)
          pass variables:
          lines       = 126
          density     = density
```