The Space Articulation Network Reduction Heuristic

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 11-08
November 1, 2011

Last revision: November 3, 2011.

1. INTRODUCTION

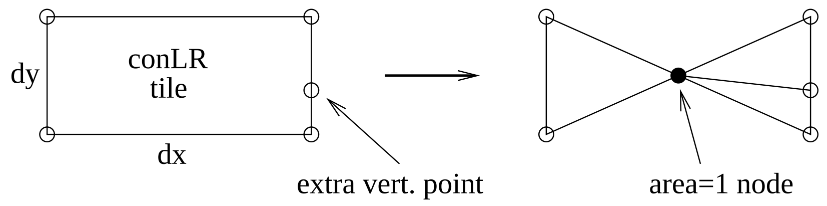
When doing a *space* resistance extraction, the nodes are classified with the **area** flag for articulation reduction. The **area** flag can have three values:

```
    0 = non equi-potential node
    1 = equi-potential line node
    2 = equi-potential area node
```

The **area** flag is set with functions *makeLineNode* and *makeAreaNode* in the code. Also the number of sets is counted with variables *equiLines* and *areaNodes* for statistics. To print these statistics, use *space* option **-i** on the command line. The variable *areaNodesTotal* is used to count the "total ready area nodes". This counting is done in function *readyNode* by testing the node **area** flag. The number of area nodes in a ready group can be printed with parameter **debug.ready_group**. The number is only printed for nodes w/o set **term** and/or set **keep** flag.

What is an equi-potential line node?

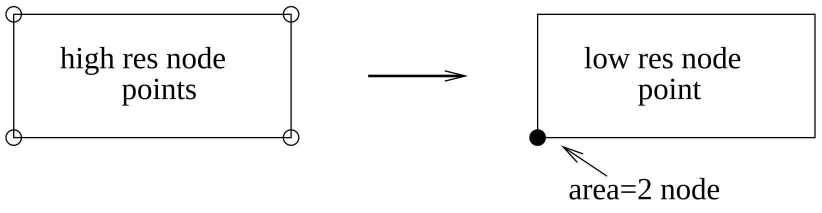
If we look where and when function *makeLineNode* is called, we see that it is called in function *doEquiRectangle* when variable *equi_line_area* is set and only for a high res conductor. Note that **equi_line_area** is a parameter, which is default "on". However, note that function *doEquiRectangle* is only called for rectangle shaped tiles and possibly only when parameter **equi_line_ratio** is greater than zero (default "0"). Function *doEquiRectangle* is only called when the tile is a only **conTB** or only **conLR** type tile. Note that the **conTB** tile may not have extra points on the vertical tile edges and the ratio dy/dx must be >= **equi_line_ratio**. Note that the **conLR** tile may not have extra points on the horizontal tile edges and the ratio dx/dy must be >= **equi_line_ratio**. Function *doEquiRectangle* creates an equi-potential line node in the center of the tile, see example:



The tiles are classified **conTB** in function *resEnumPair* when a horizontal edge crossing contains on both sides the same high res conductor. And a tile with a high res d/s-area is also classified **conTB** by a horizontal edge crossing between gate and d/s-area.
The tiles are classified **conLR** when a vertical edge crossing contains on both sides the same high res conductor. And a tile with a high res d/s-area is also classified **conLR** by a vertical edge crossing between gate and d/s-area.

What is an equi-potential area node?

When we look in the code and look where function *makeAreaNode* is called. We see that function *makeAreaNode* is called in functions *doRectangle* and *doEquiRectangle* and only is called for low res conductor subnodes. These functions are called by function *triangular*. And *triangular* is called by function *resEnumTile*, but only if the tile has a high res conductor. Thus, function *triangular* is not called for a completely low res conductor. Tile example:



Notes about function reducArtDegree

See for an explanation of the **min_art_degree** and **min_degree** heuristic also section 2.8.1 of the "Space User's Manual". The sub-function *calc_art_degree* is used to make the code of function *reducArtDegree* more easy to understand. The code tries to calculate the art. degree for an **area**=2 node by breaking the resistance graph in separate pieces. It uses the node **flag** to skip non-separate nodes. The function does not take a possible substrate resistor connection into account.

```
int calc_art_degree (node_t *n, int cx)
{
    degree = 0;
    for (con = n -> con[cx]; con; con = NEXT(con, n)) {
        on = OTHER(con, n);
        on -> flag = 1;
    }
    for (con = n -> con[cx]; con; con = NEXT(con, n)) {
        on = OTHER(con, n);
        if (on -> flag) { on -> flag = 0;
            ++degree;
            for (r = on -> con[cx]; r; r = NEXT(r, on)) {
                obn = OTHER(r, on);
                if (obn -> flag) obn -> flag = 0;
            }
        }
    }
    return degree;
}
```

On the following page you can find the code of function *reducArtDegree* and some comments and notes.

```
int reducArtDegree (node_t **qn, int n_cnt)
{
    for (i = 0; i < n_cnt; i++) {
        n = qn[i];
        if (n -> term > 1) { n -> keep = 1; continue; }
        n -> keep = 0;
        cx = testRCelim (n);
        if (!n -> keep && artReduc && cx >= 0) {
            if (n -> area) {
                if (n -> area == 1) degree = 2;
                else  degree = calc_art_degree (n, cx);
                if (degree >= min_art_degree || (degree > 1 &&
                  n -> res_cnt >= min_degree)) n -> keep = 1;
            }
            else if (n -> term) {
                if (min_art_degree <= 1) n -> keep = 1;
            }
        }
        if (!n -> keep && !n -> term) {
            elim (n);
            qn[i--] = qn[--n_cnt];
        }
    }
    return (n_cnt);
}
```

Some comments about the code of the above *reducArtDegree* function. Variable *artReduc* can only be true by resistance extraction and is only true if one (or both) of the parameters **min_art_degree** or **min_degree** is >= 0 and less than MAX_INT. Both parameters are default MAX_INT (a specified value < 0 is changed into MAX_INT).

You see that the value of both parameters can set the node **keep** flag, thus the node will be retained in the network. Note that a **term** node is always retained in the network, but the **keep** flag can also be set for a term=1 node (which is not an **area** node). However, this shall normally not happen, because **min_art_degree** <= 1 is normally not used.

Note that the **keep** flag is only used in function *reducMinRes* for the **min_res** heuristic. Nodes with a set **keep** flag are not evaluated by function *reducMinRes*. But function *reducMinRes* shall only evaluate nodes with an unset **keep** flag and that can only be nodes with **term**=1 status.

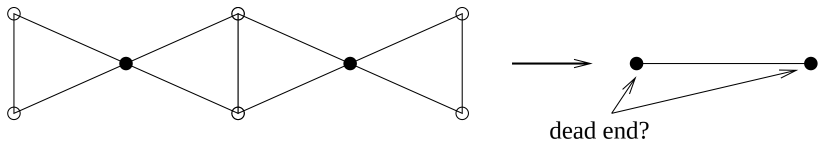
Note that **cx** >= 0 guarantees that there is a resistor connected to node **n**. And, because the **keep** flag is not set, there is only one resistor type connected to the node. However, the node **res_cnt** can be zero, because the resistor can be a substrate resistor. Note that the substrate resistor is not taken into account when calculating the art. degree. In fact the test for **cx** >= 0 is incorrect, because now not always the **keep** flag can be set for all **area** nodes when parameter **min_art_degree** = 0 is specified.

Note that the use of an art. degree of 2 for **area**=1 nodes does not follow the specification of section 2.8.1 in the "Space User's Manual". It can better be coded like this:

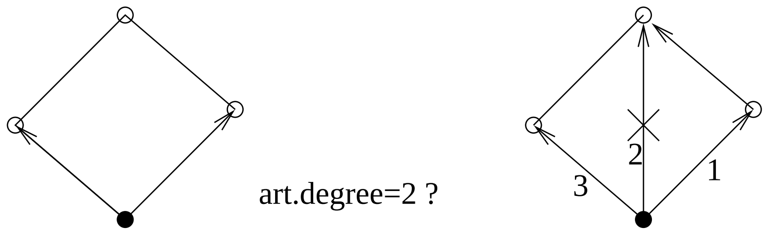
```
    if (n -> area == 1) {
        if ((degree = n -> res_cnt) > 2) degree = 2;
    }
```

When the suggested (typical) settings are used, **min_art_degree**=3 and **min_degree**=4, then the **keep** for **area**=1 nodes is only depended of the node **res_cnt** and the choice of parameter **min_degree**.

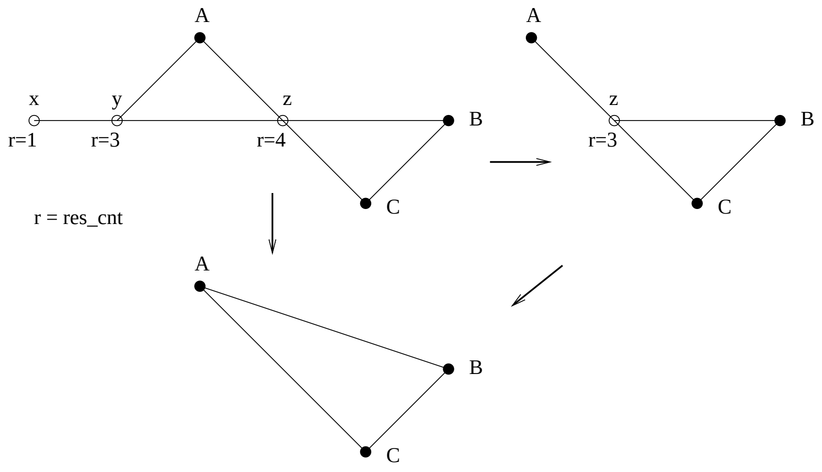
By using a fixed art. degree of 2 for **area**=1 nodes and using **min_art_degree**=2, then all **area**=1 nodes will be retained in the network and possible also nodes on a dead end (dangling nodes). See following figure:



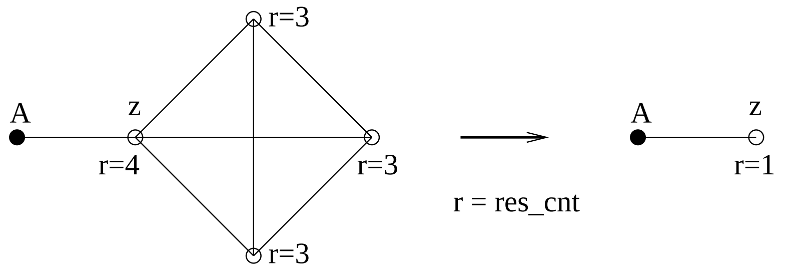
Note that function *calc_art_degree* does not compute the **real** articulation degree. It does not take interconnect loops into account and does not find an art. degree of 1 for the following cases:



When we have three line nodes (x,y,z), then the order of processing of the nodes decide of line node 'z' is eliminated or not. Only when node 'z' is done after 'x' and 'y', it is eliminated. When node 'y' is done before 'x', then the elimination is more complex. We can add parameter **art_reduc_retry** to process node 'z' a second time and to eliminate it.



When line node 'z' is done before the other line nodes (and no retry is done), then it is possible that node 'z' becomes a dead end (a dangling node). See figure:



The parameter **delete_dangling** is added to the code of function *reducArtDegree* to eliminate dangling nodes. Note that a dangling line node is also not eliminated when **min_art_degree** is <= 2. Thus **delete_dangling** must overrule the art. parameters.
This gives the following code:

```
int reducArtDegree (node_t **qn, int n_cnt)
{
    for (i = 0; i < n_cnt; i++) {
        n = qn[i];
        if (n -> term > 1) { n -> keep = 1; continue; }
        n -> keep = 0;
        cx = testRCelim (n);
        if (!n -> keep && artReduc && cx >= 0) {
            if (n -> area) {
                if (delete_dangling && n -> res_cnt < 2 && !n -> term
                  && (n -> substrCon[cx] == 0 || n -> res_cnt == 0)) goto E1;
                if (n -> area == 1) degree = 2;
                else  degree = calc_art_degree (n, cx);
                if (degree >= min_art_degree || (degree > 1 &&
                  n -> res_cnt >= min_degree)) n -> keep = 1;
            }
            else if (n -> term && min_art_degree <= 1) n -> keep = 1;
        }
        if (!n -> keep && !n -> term) {
E1:         if (delete_dangling && i && n -> res_cnt <= 2) {
                for (con = n -> con[cx]; con; con = NEXT(con, n)) {
                    on = OTHER(con, n);
                    if (on -> res_cnt <= 2 && !on -> term) {
                        for (j = 0; j < i && qn[j] != on; ++j);
                        if (j < i) { qn[i--] = on; qn[j] = qn[i]; }
                    }
                }
            }
            elim (n); qn[i--] = qn[--n_cnt];
        }
    }
    return (n_cnt);
}
```

We must look out for **cx** < 0 in the second delete_dangling test?

The evolution of the reducArtDegree source code

The order problem was no issue in early source code of function *reducArtDegree* (source delta 4.10 to 4.14 in 1993). Function *reducArtDegree* did not yet exist, but was part of function *reducGroupI*. See following code part:

```
if (optRes && (min_art_degree >= 0 || min_degree >= 0)) {
   for (i = 0; i < n_cnt; i++) {
      n = qn[i];
      n -> help = 0;  /* the articulation degree */
      n -> keep = 0;
      cx = testRCelim (n); /* substrCon not tested! */
      if (!n -> keep && cx >= 0) {
         for (con = n -> con[cx]; con; con = NEXT(con, n)) {
            on = OTHER(con, n);
            if (!on -> flag) { setBranch (n, on, cx); n -> help += 1; }
         }
         for (j = 0; j < n_cnt; j++) qn[j] -> flag = 0; /* reset */
      }
   }
   for (i = 0; i < n_cnt; i++) {
      n = qn[i];
      if (!n -> keep) {
         if ((min_art_degree >= 0 && n -> help + 0.5 >= min_art_degree)
         || (n -> help > 1.5 && n -> res_cnt >= min_degree)) n -> keep = 1;
      }
      if (n -> area && !n -> keep) { elim (n); qn[i--] = qn[--n_cnt]; }
   }
}
```

Note that there are not yet substrate resistors. The **flag** member of all nodes is initial 0. Function *setBranch* sets recursively node flags and counts the number of branches for node 'n'. Thus, the **real** articulation degree is computed and stored in the node **help** member (a double). This is done in the first for-loop for all the nodes, which don't need to be kept. When **cx** is >= 0, we know that there is a resistor connected to node 'n' and that an art. degree for node 'n' must be computed. And all node flags are reset again.
The second for-loop shall evaluate the computed art. degree of the nodes and can set **keep** depended of **min_art_degree** and/or **min_degree** parameters. The **area** nodes with unset **keep** flag are eliminated. Note that terminal nodes must not be area nodes.

Note that an **area** node is always kept when **min_art_degree**=0.

Starting with revision 4.15 (1993-09-20) the second for-loop is not more used. From that moment on, the processing order of the nodes become important, because some nodes may be eliminated during the art. reduction process. See code on next page.

```
if (optRes && (min_art_degree >= 0 || min_degree >= 0)) { /* >= 4.15 */
   for (i = 0; i < n_cnt; i++) {
      n = qn[i];
      n -> keep = 0;
      cx = testRCelim (n); /* substrCon not tested! */
      if (n -> area && !n -> keep && cx >= 0) {
         n -> help = 0;  /* the articulation degree */
         for (con = n -> con[cx]; con; con = NEXT(con, n)) {
            on = OTHER(con, n);
            if (!on -> flag) { setBranch (n, on, cx); n -> help += 1; }
         }
         for (j = 0; j < n_cnt; j++) qn[j] -> flag = 0; /* reset */

         if ((min_art_degree >= 0 && n -> help + 0.5 >= min_art_degree)
         || (n -> help > 1.5 && n -> res_cnt >= min_degree)) n -> keep = 1;
      }
      if (n -> area && !n -> keep) { elim (n); qn[i--] = qn[--n_cnt]; }
   }
}
```

Note that this version depends on **cx** and does not always keep an **area** node when **min_art_degree**=0. Note that the node **help** member needs not more be used for the art. degree. Revision 4.19 (1993-12-23) adds the test (min_degree >= 0) before (n -> help > 1.5) and removes the test for (n -> area) to compute the art. degree. In that case the art. degree is also computed for terminal nodes and **keep** possibly set.

Starting with revision 4.22 (1994-08-05) a separate *reducArtDegree* function is created. Now the art. degree is only computed for **area** nodes, for terminal nodes a degree of 1 is used. The node **term** member is used in some tests, but the code is not really different.

Starting with revision 4.28 (1996-08-15) some major changes can be reported. Now area line nodes are introduced, which have a fixed art. degree of 2. Function *setBranch* is not more recursive and also *unsetBranch* is added to reset the **flag** node members. Thus, not more the **real** articulation degree is computed! Now also substrate extraction code is added, function *testRCelim* is now also testing for substrCon/Cap and for junction caps. Note that function *testRCelim* is really added in revision 4.29 (1998-02-06).

Starting with revision 4.45 (2004-01-16) the test for **term** > 1 is added and setting the **keep** member. Line area nodes are now changed into **area**=1 nodes.

Starting with revision 4.54 (2009-06-18) changed n -> help into local **degree** variable. Separate articulation degree tests for **area** and **term** nodes.

Starting with revision 4.55 (2009-06-28) removed functions *setBranch* and *unsetBranch*. Added the **artReduc** variable and removed the test for **optRes**, min_art_degree and min_degree. No separate code anymore for the not **optRes** mode.

Starting with revision 4.59 (2010-05-04) removed tests for **min_art_degree** >= 0 and **min_degree** >= 0. In revision 4.62 (2010-12-14) removed the test for n -> substr.
In revision 4.67 (2011-10-27) added **delete_dangling** code.

How to implement an efficient art_reduc_retry?

We can easy test on the end of the for-loop of some nodes are eliminated and then jump back and start function *reducArtDegree* again. A jump back is not needed, if only nodes on i=0 are eliminated. See following code:

```
int reducArtDegree (node_t **qn, int n_cnt)
{
begin_art:
    retry = 0;
    for (i = 0; i < n_cnt; i++) {
        n = qn[i];
        if (n -> term > 1) { n -> keep = 1; continue; }
        n -> keep = 0;
        cx = testRCelim (n);
        if (!n -> keep && artReduc && cx >= 0) {
            if (n -> area) {
                if (delete_dangling && ...) goto E1;
                ...
            }
            else if (n -> term) {
                if (min_art_degree <= 1) n -> keep = 1;
            }
        }
        if (!n -> keep && !n -> term) {
E1:         if (i) {
                if (art_reduc_retry) retry = 1;
                else if (delete_dangling) { ... }
            }
            elim (n);
            qn[i--] = qn[--n_cnt];
        }
    }
    if (retry) goto begin_art;
    return (n_cnt);
}
```

Maybe we don't need to start over again by i=0, for example when the first nodes in array qn have the term > 1 status. Possibly we can also swap nodes to the begin of qn?

A more efficient calc_art_degree

The code to compute the art. degree for **area**=2 nodes can be improved:

```
if (n -> area == 2) {
    degree = calc_art_degree (n, cx);
    if (degree >= min_art_degree ||
       (degree >= 2 && n -> res_cnt >= min_degree)) {
        n -> keep = 1;
        continue;
    }
}
```

We can change it into:

```
if (n -> area == 2) {
    degree = calc_art_degree (n, cx);
    if (degree < min_art_degree &&
       (degree < 2 || n -> res_cnt < min_degree)) goto nokeep;
    n -> keep = 1;
    continue;
}
```

And we can change it into:

```
if (n -> area == 2) {
    if (min_art_degree < 2) {
        if (n -> res_cnt < min_art_degree) goto nokeep;
    }
    else {
        min = (n -> res_cnt < min_degree) ? min_art_degree : 2;
        if (n -> res_cnt < min) goto nokeep;
        degree = calc_art_degree (n, cx);
        if (degree < min) goto nokeep;
    }
    n -> keep = 1;
    continue;
}
```

Thus, by a **min_art_degree** < 2 we don't need to compute the degree.
By a **min_art_degree** >= 2 we only need to compute the degree by a **res_cnt** >= min.
This, because the art. degree is always <= **res_cnt**.