The Space max_par_res 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 11-09
November 14, 2011

Last revision: November 21, 2011.

1. INTRODUCTION

When doing a *space* resistance extraction, the **max_par_res** heuristic prevents the occurrence of high-ohmic shunt paths between two nodes. If the ratio of the absolute value of a resistor and its minimum parallel resistance path (along positive resistors) exceeds the value of the **max_par_res** parameter, then the high-ohmic shunt resistor is simply removed. This is done in function *reducMaxParRes*. See following code fragment:

```
int reducMaxParRes (node_t **qn, int n_cnt)
{
   if (max_par_res >= 1 && n_cnt > 1) {
      for (i = 0; i < n_cnt; i++) /* reset flag2, flag3 */
      area_nodes = 0;
      for (i = 0; i < n_cnt; i++) { /* Evaluation loop(1) */
         n = qn[i]; if (n -> flag2) continue;
         if (n -> area) { ++area_nodes; continue; }

         QC_cnt = 0; /* make clique for n */
         putQueueClique (n); /* set flag3 */
         for (k = 0; k < resSortTabSize; k++)
         for (con = n -> con[k]; con; con = NEXT (con, n)) {
            if (con -> type == 'S') continue;
            on = OTHER (con, n); /* Grp(on) == Grp(n) */
            if (!on -> flag3) putQueueClique (on);
         }

         /* (1) Evaluate resistors between all pairs of nodes
         for (l = 0; l < QC_cnt-1; l++) ...  in the clique */

         for (l = 0; l < QC_cnt; l++) /* set flag2; reset flag3 */
      }
      if (area_nodes < 2) return (n_cnt);

      /* (2) Cliques with only area nodes are still uninvestigated now */
      QC_cnt = 0;
      for (i = 0; i < n_cnt; i++) /* make clique */
         if (qn[i] -> area) putQueueClique (qn[i]); /* set flag3 */

      /* (2) Evaluate resistors between all pairs of area nodes
      for (l = 0; l < QC_cnt-1; l++) ... in the clique */
   }
   return (n_cnt);
}
```

Node **flag2** is only used in function *reducMaxParRes*, in Evaluation loop(1), to flag that a not area node is already done in a clique.

Node **flag3** is used in function *reducMaxParRes* to flag that the node is in the clique and that the res path must be evaluated to current node 'n'. When set to 2, the clique node is already done (see next code part). Node **flag3** is also used in function *min_R_path* to flag that the node is in the clique.

For-loop (1) evaluates the resistors between all pairs of nodes in the clique using following code:

```
for (l = 0; l < QC_cnt-1; l++) {
   n = QC[l]; n -> flag3 = 2; /* done */
   for (j = 0; j < QC_cnt; j++) QC[j] -> help = -1; /* init */

   r_flag = 0; /* used to speed-up the searching */
   for (k = 0; k < resSortTabSize; k++)
   for (con = n -> con[k]; con; con = NEXT (con, n)) {
      if (con -> type == 'S') continue;
      on = OTHER (con, n); /* Grp(on) == Grp(n) */
      if (on -> flag3 == 1) {
         if ((res = 1 / con -> val) < 0) res = -res;
         min = min_R_path (n, on, QC_cnt, r_flag);
         if (min > 0 && min * max_par_res < res) elemDel (con);
         r_flag = 1;
      }
   }

   for (j = 0; j < QC_cnt; j++) QC[j] -> flag = 0; /* reset */
}
```

Note that the test for a type 'S' resistor is not really needed when **flag3** can only be set for a node in the current node group. This is also true for function *min_R_path*. Note that better **flag** can be used, because the status for **flag** is guaranteed initial zero. Maybe **flag3** and **flag** can be combined. Maybe also **flag2** can be combined with **flag**.

Function *min_R_path* uses three node members, (1) **help** (initial -1) to set the minimum res path for a clique node to node 'n' and (2) **flag** (initial 0) to flag a clique node that the minimum path is found and (3) **flag3** to identify a clique node. The clique node (!= n) can become a front node of node 'n', when a res path to node 'n' is set in the **help** member. Note that the **help** member could also be inited to 0 (in place of -1). In that case it must be sure that **help** is always set in *min_R_path* to a value > 0.

See the code of function *min_R_path* on next page.

Note that for each 'n' also the resistor to 'ns' is done, because 'ns' **flag3**=2. But because 'ns' **help**=0 it is not doing "if(on->help < 0)" and "if(min < on->help)".

Note that function *findMaxRes* is also using function *min_R_path*, but only when stream "wiredata" must be written. This happens only for the "inspector" version of *space*.

Note that function *findMaxRes* does not use the clique array (QC and QC_cnt), it uses the **qn** array. The clique array **QC** is only needed in Evaluation loop(1).

```
double min_R_path (node_t *ns, node_t *nt, int n_cnt, int doContinue)
{
   static int f_cnt, min_R_tabsize = 0;
   static node_t *n, **frontNodes;

   if (!doContinue) { n = ns; f_cnt = 0; n -> help = 0;
      if (n_cnt > min_R_tabsize) /* alloc space for frontNodes */
   }
   else if (nt -> flag) return (nt -> help); /* previous result? */

   while (n && n != nt) {
      for (k = 0; k < resSortTabSize; k++)
      for (con = n -> con[k]; con; con = NEXT (con, n)) {
         if (con -> type == 'S') continue;
         on = OTHER (con, n); /* Grp(on) == Grp(n) */
         if (on -> flag3 && con -> val > 0) { /* clique node */
            min = 1 / con -> val + n -> help; /* min > 0 */
            if (on -> help < 0) {
               frontNodes[ f_cnt++ ] = on; on -> help = min;
            }
            else if (min < on -> help) on -> help = min;
         }
      }

      if (f_cnt > 0) {
         min = frontNodes[f=0] -> help;
         for (i = 0; ++i < f_cnt;)
            if (frontNodes[i] -> help < min) min = frontNodes[f=i] -> help;
         n = frontNodes[f];
         frontNodes[f] = frontNodes[--f_cnt];
         n -> flag = 1; /* minimum res path found */
      }
      else n = NULL;
   }
   return (nt -> help); /* if(help < 0) no res path */
}
```

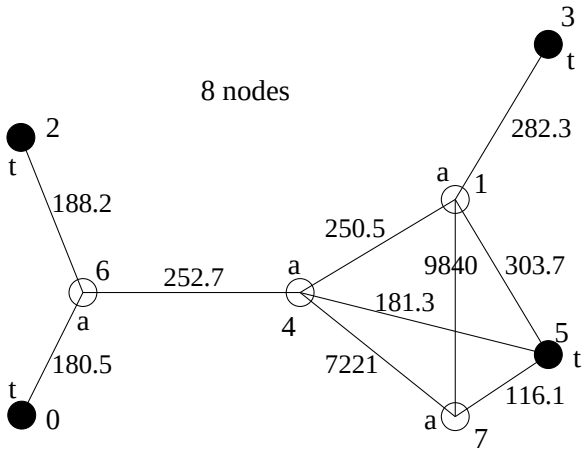
The **help** member in the nodes contain the found res path for node 'ns'. The goal is to find the res path to target node 'nt'. When all front nodes are done and no front node is 'nt', then 'n' becomes NULL and 'nt' **help** is not set.
Note that the **help** members of the nodes don't need to be inited at all. We can use for the status of the nodes the **flag** member. Only the first node 'n' needs to be initialized. The returned res path is always the **help** of one of the front nodes (or else -1).

The *doContinue* variable can be used to find the res path for another front node 'nt' for node 'ns'. Because at the end, each front node (= clique node) must contain the minimum res path till node 'ns' (or there was no res path along positive resistors to the node 'nt').

Note that the *frontNodes* array is not needed, we can also use the node **next** pointers instead.

2. EXAMPLE

Given the following resistor network with 8 nodes:



Function *reducMaxParRes* can now try to remove too high-ohmic shunt resistors. It starts in Evaluation loop(1) with the non area nodes. It starts with node 0, this node has only one resistor and does not really to be done. Nodes 2 and 3 have also only one resistor. Node 5 is the last candidate to be node 'n' and to make a clique. Nodes 1, 4 and 7 are placed with node 'n' in the clique (in random order).

We start with node 5 to become 'ns' in function *min_R_path*. For all target nodes 'nt' we find minimum res paths, which are equal to the resistor values connected to node 5. We start again with a new node 'ns', for example node 4. Now, the min res path to node 5 does not more to be done. We see that resistor 7221 has a smaller res path to node 7 (181.3 + 116.1). By a **max_par_res** ratio of 20, we can remove the resistor of 7221 ohm. When we start again with node 7, we don't need to do the min res path to node 5. We see that resistor 9840 has a smaller res path to node 1 (116.1 + 303.7), thus also this resistor can be removed.

We see that node 7 can become a *dangling* local node (if it is no terminal).

A second evaluation loop is done for all area nodes (if there are more than one). This is just in case there are cliques with only area nodes, which are uninvestigated. Note that the area nodes are all placed in one clique. In this example there are four area nodes, but no new reductions can be done.

Note that, when a resistor can be removed, we maybe better can re-evaluate both nodes. Maybe one of the nodes can be eliminated because of a becomming too low art. degree. In that case the resistor does not need to be removed, because the elimination changes the resistor network.