Space SNE reduction notes

S. de Graaf

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

Report EWI-ENS 11-05
August 15, 2011

Copyright © 2011 by the author.
All rights reserved.

Last revision: September 27, 2011.

1. INTRODUCTION

Selective Node Elimination (SNE) a method for frequency depended network reduction. The SNE method works only when using the *space* resistance and capacitance extraction mode and using the **-G** option for SNE. Nevertheless you can fool the *space* extractor by not specifying capacitances in the technology file or using a high "low_sheet_res" and "low_contact_res" parameter for resistances. But note, that SNE can only work for nodes which have both resistors and capacitors.

The SNE method uses default a frequency of 1e9 Hz, which can be changed with parameter "sne.frequency". Other important parameters are: sne.norm (default 0), sne.errorfunc (default 0), sne.fullgraph (default 0), sne.tolerance (default 0.05), sne.resolution (default 1) and moments.max (default 2).

The SNE method works by calculating a weight for delayed[1] nodes. Based on the weight the nodes get higher priority and are not as easy eliminated. Based on the "sne.resolution", each priority queue degree entry is split in separate sections. When the weight reaches the "sne.tolerance" value, it is placed in the highest section and can at that moment not be eliminated. A weight value is default calculated by only evaluating the grounded branches of the node (because of sne.fullgraph=0) and only the highest value of one branch is taken (because of sne.norm=0) and only the first extra moment is used (because of sne.errorfunc=0). Therefor using a higher "moments.max" has no effect and default SNE can only work with "moments.max=2". Only "sne.norm=3" method has default "moments.max=0", because moments are not used.

The procedure in the *space* program works as follows. Each ready node is processed by function readyNode. When it is the last node of the node group, it is given to function readyGroup. When it is not the last node and can be delayed, it is given to function nqDelayElim. Function nqDelayElim places the node in the priority queue by calling nqInsert. But first the node "degree" is calculated by nqDegree and for SNE the node "weight" is calculated by updateWeight (node is also flagged as delayed). Function nqDelayElim counts the number of delayed nodes, when too many nodes are delayed (parameter max_delayed), then one of the nodes in the priority queue is eliminated. Normally the value of parameter max_delayed is very large, thus this happens not easy. More likely a node group becomes completely ready and last node is given to readyGroup. If possible, this last node is delayed by readyGroup. Now all delayed nodes of the group are eliminated by function nqEliminateGroup. Each eliminated node updates the weights of its connected nodes. However, delayed nodes which get enough weight, are not eliminated. Use parameter "sne.print_elimcount" to see info.

---

1. Delayed nodes are nodes that can be eliminated. They are placed in a priority queue. Nodes with lowest degree are first eliminated. Area and terminal nodes are not delayed.

2. WEIGHT CALCULATION

The ready delayed nodes get an initial weight, using function updateWeight. Default (sne.fullgraph=0) only the weight for the grounded branches is calculated and only the larged weight found (sne.norm=0) is taken. The node must have a grounded branch else no weight can be calculated. Therefor, the node must have a groundCap or a groundCap moment value. Using moments.max=2 the following formula are used for the RES (and CAP) situation:

```
Mij_[0] = (Gi*Gj) / GS
Mij_[1] = (Ci*Gj + Gi*Cj - Mij_[0]*CS) / GS
Mij_[2] = (Ci*Cj + Mi*Gj + Gi*Mj - Mij_[0]*MS - Mij_[1]*CS) / GS
```

For RES and only i=0 (Gi=0, Ci=groundCap, Mi=moment) the formula are:

```
Mij_[0] = 0
Mij_[1] = (Ci*Gj) / GS
Mij_[2] = (Ci*Cj + Mi*Gj - Mij_[1]*CS) / GS
```

For NORES (the only CAP situation) the following formula are used:

```
Mij_[0] = 0
Mij_[1] = (Ci*Cj) / CS
Mij_[2] = (Ci*Mj + Mi*Cj - Mij_[1]*MS) / CS
```

Initial there can't be moments, thus Mi=0, Mj=0 and MS=0. And there can only be a branch weight for sne.errorfunc=0, if Mij_[2] != 0. Thus, there is only a branch weight in the RES/CAP situation with Mij_[2] formula:

```
Mij_[2] = (Ci*Cj - Mij_[1]*CS) / GS
Mij_[2] = (Ci*Cj - Ci*Gj*CS/GS) / GS
```

Note that Cj or Gj (Mij_[1]) can be zero (but not both) (Ci, CS and GS are not zero).
For sne.norm=0 and sne.errorfunc=0 the calculated branch weight for branch(j) is:

```
weight(j) = sneOmega * abs( Mij_[2] / ( Mij_[1] + CGj ) )
weight(j) = sneOmega * abs(Ci*Cj - Mij_[1]*CS) / (GS*(Mij_[1] + CGj))
```

For CGj=0 branch Gj (Mij_[1]) may not be zero (else weight(j)=0):

```
weight(j) = sneOmega * abs(Ci*Cj - Mij_[1]*CS) / (GS*Mij_[1])
weight(j) = sneOmega * abs(Cj/Gj - CS/GS)
```

For Gj=0 branch CGj may not be zero (else weight(j)=0):

```
weight(j) = sneOmega * abs((Ci*Cj) / (GS*CGj))
```

For this last situation, a weight update of a node over a coupleCap Cj is only useful, when both nodes have a groundCap.

3. MOMENTS CALCULATION

Extra moments are only calculated by function momentsElim and only when maxMoments > 1. Function momentsElim is only called by function elim. Variable maxMoments is default 2 (parameter moments.max=2). This results in the calculation of 1 extra moment. This is for the SNE method enough (for sne.errorfunc < 2). The value of the extra moment is stored in a capacitor element, if the capacitor is a coupleCap between two nodes. When the capacitor is a groundCap, than there is no capacitor element, but the groundCap value is stored in the node. Thus, in that case the extra moment value is also stored in the node. When the capacitor is a substrateCap, than the values are also stored in the node.

Extra moments can only exist when one or more nodes are eliminated. When a node group becomes ready and there are not yet nodes of the group eliminated, then there are not yet extra moments. However, coupleCap elements which connect the group with other adjacent ready groups have most likely an extra moments value, because delayed nodes (if existing) of the adjacent groups are most likely eliminated.

Function momentsElim can only calculate an extra moment if the eliminated node has a capacitor or a groundCap moment value (but initial a groundCap moment value can not exist).
For RES (and CAP) the formula are:

```
Mij_[0] = (Gi*Gj) / GS
Mij_[1] = (Ci*Gj + Gi*Cj - Mij_[0]*CS) / GS
Mij_[2] = (Ci*Cj + Mi*Gj + Gi*Mj - Mij_[0]*MS - Mij_[1]*CS) / GS
```

For RES (no CAP) the formula are:

```
Mij_[1] = 0
Mij_[2] = (Mi*Gj + Gi*Mj - Mij_[0]*MS) / GS
```

For RES and no moments (there must be CAPs else Mij_[2] = 0), formula:

```
Mij_[2] = (Ci*Cj - Mij_[1]*CS) / GS
```

For NORES (only CAPs) the formula are:

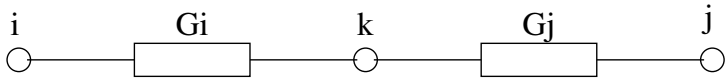
```
Mij_[0] = 0
Mij_[1] = (Ci*Cj) / CS
Mij_[2] = (Ci*Mj + Mi*Cj - Mij_[1]*MS) / CS
```

For NORES and no moments: Mij_[2] = 0.
Thus, the eliminated node must have RES and CAP to calculate an extra moment.
Below is explained what the calculated moments are:

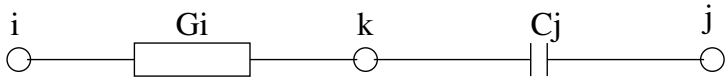
```
Mij_[0] = the calculated conductance (Gij) component
Mij_[1] = the calculated capacitance (Cij) component
Mij_[2] = the calculated extra moment(Mij) component
```

The following figure summarize the new component that is added between node i and j.

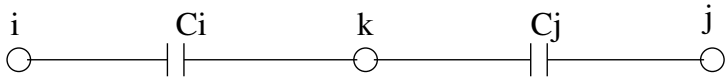
The figure below gives an overview of the calculated additional component between node i and j by elimination of node k in a RES/CAP situation. GS is the sum of the conductances and CS is the sum of the capacitances connected to node k. An extra moment Mij is calculated in a RES/CAP situation (CS != 0).



$$Gij = Gi * Gj / GS$$
$$Cij = -Gij * CS / GS$$
$$Mij = -Cij * CS / GS$$

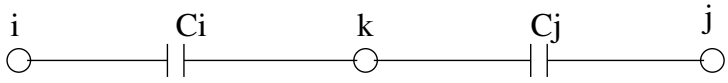


$$Gij = 0 \qquad Cij = Gi * Cj / GS$$
$$Mij = (Gi * Mj - Cij * CS) / GS$$



$$Gij = 0 \qquad Cij = 0$$
$$Mij = Ci * Cj / GS$$

The figure below gives an overview of the calculated additional component between node i and j by elimination of node k in a only CAP situation (GS = 0). MS is the sum of the (extra) moments connected to node k.

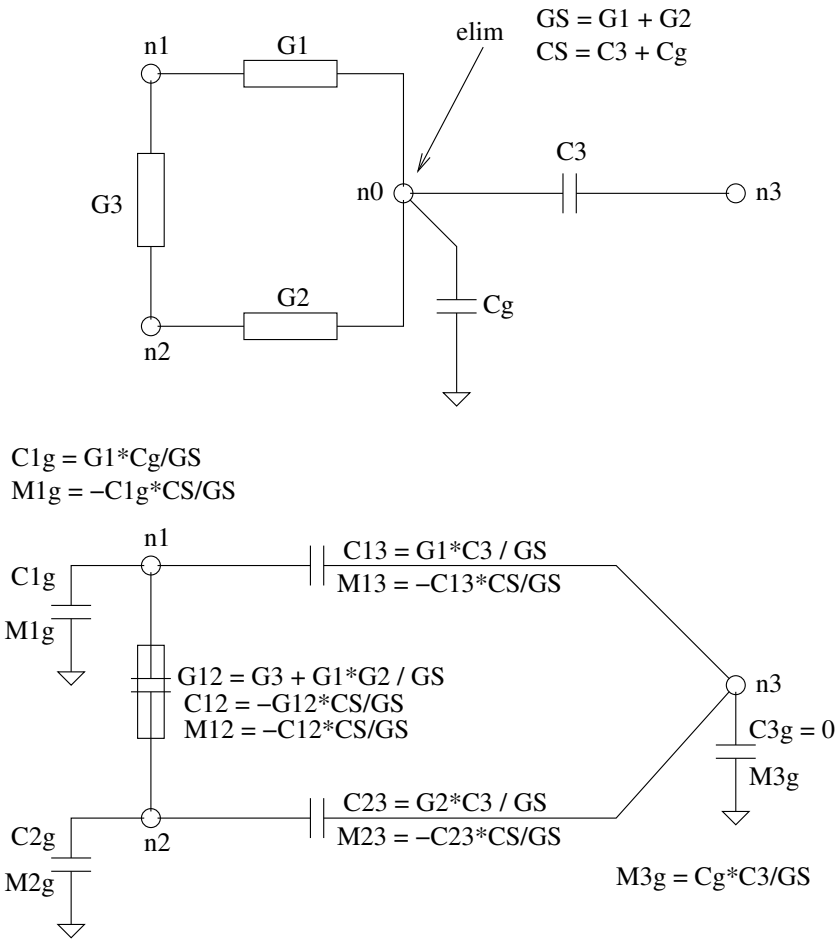


$$Gij = 0 \qquad Cij = Ci * Cj / CS$$
$$Mij = (Ci*Mj + Mi*Cj - Cij*MS) / CS$$

Thus a only CAP situation needs a RES/CAP situation, because else no moment Mij can be calculated.

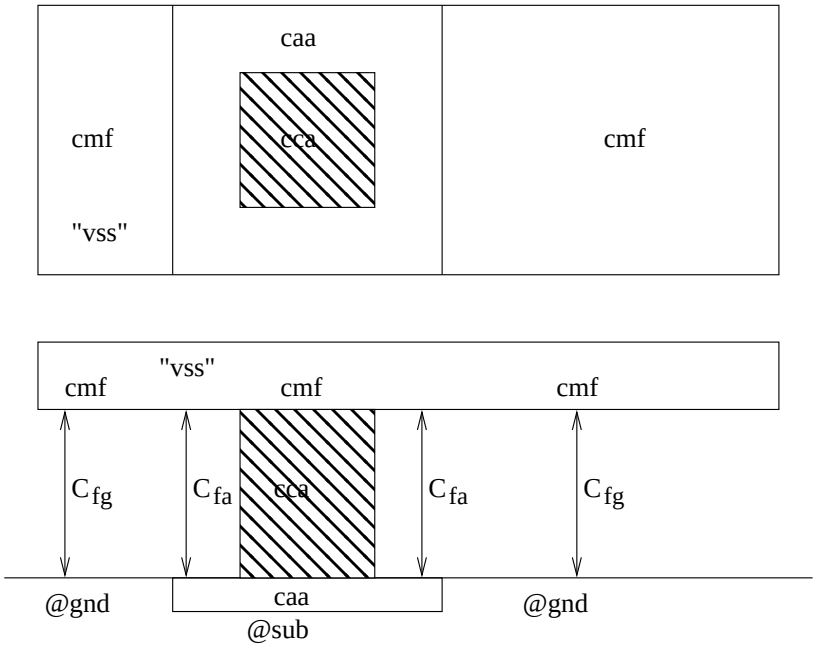
The figure on the following page gives an example how extra moments are created by node elimination.

The figure below shows a node elimination and the creation of moments.



When nodes n0, n1 and n2 are in another group than node n3, you see that the elimination of node n0 has impact on the node n3 of another group. That node gets more coupleCap elements and gets a groundCap. Note that this groundCap has only an extra moments value. Thus a grounded branch for n3 is created via a coupleCap to another group.

As an example for SNE we shall look to an active area part on the "vss" supply line which is used for substrate contacts in the "oscil" demo example (see figure 1 below).

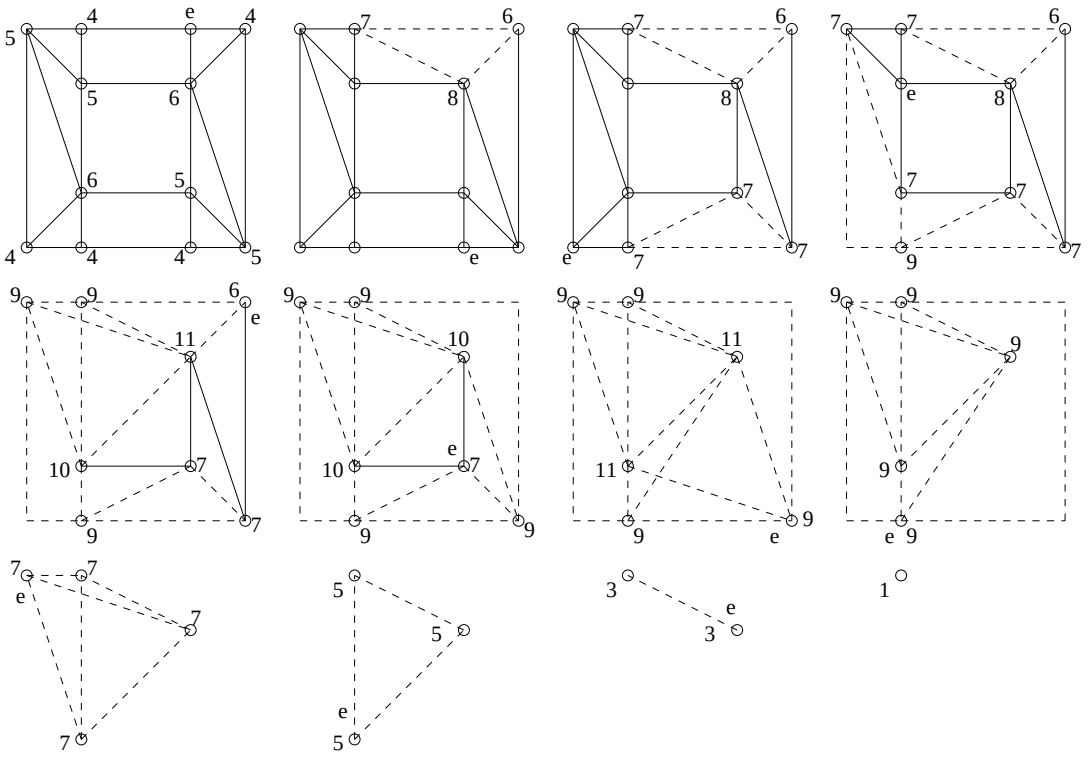



```
cont_b : cca  cmf  !cwn  !csn  :  cmf  @sub  : 80
acap_cmf_sub : cmf  !cpg  !caa  !cwn : cmf  @gnd  : 25 # Cfg
acap_cmf_caa : cmf  !cpg   caa  !cca   : cmf   caa  : 49 # Cfa
```

The substrate contact "cont_b" (see above) is defined between first metal (cmf) and substrate (@sub), and layer "cca" is used as the contact mask. Also the active area mask (caa) is used for the contact, but not in the "cont_b" definition. This active area part is not connected with the substrate and not connected with "cmf" and not connected with the touching "caa" d/s area (because it is of another type). And this "caa" part is an isolated high resistive island. Between the "caa" and "cmf" mask a couple cap is defined (Cfa), which is different to the couple cap between "cmf" and the substrate or ground @gnd (Cfg). Note that if the "caa" island is removed, the couple cap definition "acap_cmf_sub" needs the additional condition "!cca".

When extracting, the *space* extractor uses a scanline technique and the active area island is split in 5 tiles. These tiles get nodes on the corners (12 in total) and a resistor mesh exists between these nodes (see figure 2 on the next page). There is a procedure to eliminate nodes, at last one is left. The steps are shown in the figure.
Note that each of the 12 nodes has a couple capacitor to "vss" (cmf). Because all 12 nodes are delayed, all nodes of the island are eliminated. Thus, no dangling couple capacitor of the island is left. By elimination, see the dashed lines, also a couple capacitor is created between the nodes (parallel to a resistor).

Below, figure 2 shows the resistor mesh of 12 nodes and the elimination of 11 nodes.



By SNE, the weights of connected delayed nodes are updated, but not for the "vss" node.

> Note that the degree of the nodes is also updated. Maybe, for SNE, we can better use only the resistor count for the node degree (set parameter elim_order=1).

When the nodes of the above mesh are only connected to nodes with couple caps (the *space* **-rC** mode), the weight update of the delayed nodes is for "sne.fullgraph=0" not succesful. Because these nodes need to have a grounded branch. When the first metal conductor is high res, then the "vss" node is split in a resistor mesh and a number of delayed nodes. In that case the couple caps can update the weights of the nodes of the adjacent "vss" group (succesful, because these nodes have a grounded branch).

> Note that not all adjacent group nodes are ready. In function clearTile it is better first to finish the nodes of a ready node group. This reduce the weight updates for nodes in the adjacent group. It is also better to update delayed nodes of adjacent groups only at the end of function readyGroup (and it is maybe not needed to update the nodes of adjacent groups).

When using the *space* **-rc** mode, no coupleCaps are extracted, but only groundCaps. In that case the island becomes isolated (has no adjacent groups). All the 12 delayed nodes of the island get a weight, but are all eliminated when no weight reach the sne.tolerance value. By a sne.frequency=1e13 two nodes reach the sne.tolerance value and are not eliminated. One of the nodes is, however, eliminated by function reducMinRes (in

reducGroupI). Note that function reducArtDegree can't eliminate the nodes, because of the "term=1" status and shall normally not set the "keep" status (except when the art.degree is high and it is an area node). Nodes with "keep" status can't be eliminated by function reducMinRes. After reducGroupI, 11 of the 12 nodes of the group are eliminated. Now, the last node is given to outGroup, which calls outNode to output the node. However, function outNode shall not output the node, because it has no connections and also no names (and netEquivalences). But note that the node has the "term=1" status, but has no terminal or label. You can use the following space command to see more extraction info:

```
% space -FGrc -Ssne.print_elimcount -Sdebug.ready_group=1 \
                                          -Ssne.frequency=1e13 oscil
```

By a sne.frequency of 1e14 Hz 8 nodes reach the sne.tolerance value and are not eliminated. Five of the nodes are, however, eliminated by function reducMinRes (in reducGroupI). Thus, 3 nodes are outputted by function outNode. This gives the following dangling nodes connected with resistors in the resulting network:

```
network oscil (terminal in, out, vss, vdd, sens)
{
    res 145.2163 (1, 3);
    res 116.5244 (1, 2);
    cap 46.65876e-18 (1, GND);
    res 620.1666 (2, 3);
    cap 21.6502e-18 (2, GND);
    cap 25.77104e-18 (3, GND);
    ...
```

By a sne.frequency of 1e15 Hz all nodes reach the sne.tolerance value and are not eliminated. Eleven nodes are, however, eliminated by function reducMinRes (in reducGroupI). The last node is given to outGroup, but is not outputted by function outNode. The following code is responsible for this action:

```
void outNode (node_t *n)
{
    connected = 0;
    if (n -> res_cnt > 0) connected = 1;
    if (n -> cap_cnt > 0) connected = 1;
    if (!connected && !n -> n_n_cnt && n -> term != 2) goto ret;
    ...
```

Note that the above code of function outNode is not ok, if the local node contains both a groundCap and a resistor to substrate. In that case the groundCap must be added between the substrate and ground node (except when substrate and ground are the same node).

Why giving this detailed node group example of the island of nodes. Well, the elimination of the island cost some time and by SNE also weights are possible calculated for these nodes (and cost more time). Conclusion: After evaluation of the nodes in function readyGroup, we can decide to throw away the complete group, because it is an isolated group and not important.

4. NOTES

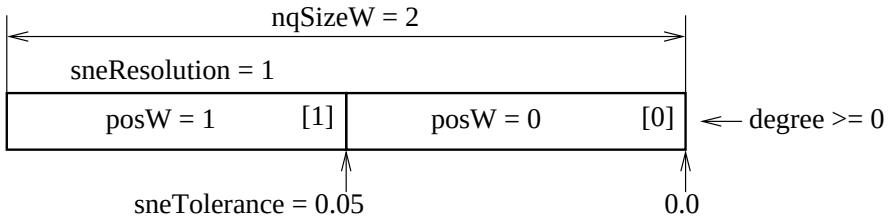
The SNE method can maybe not good work while using equi-potential line nodes. Because these nodes are not delayed and get no weight. Set parameter "equi_line_ratio=0" or don't use any heuristic (option -n). On the other hand, it is useful to have area nodes (big low res nodes), thus you don't want to put off all heuristics. This area nodes are more important and are not delayed (have no weight), but have many connections.

Maybe, in SNE mode, we must not use parameter "max_delayed" and not early eliminate ready nodes. In that case we only need to update the weight of the delayed nodes of the ready group. And that can speed-up the SNE extraction.

When extracting resistivity, only in that case there can be node groups and only in that case there can be delayed nodes. We can only use the SNE method, when there are delayed nodes.

When a node group becomes ready, we start with a set delayed nodes for SNE. We can take the set nodes apart and can calculate the initial weights or we must calculate the weight of a node if the weight of the node is reset. After calculating the weight for a node, it can be that we first must try to take another node, because of too high weight. The degree of a node has highest priority for finding a node in the priority queue. When the degree of the node is changed, the node must be repositioned in the queue. But we don't need directly to calculate a new weight value, but we must reset the value to the reset value (for example -1). A normal weight value is always >= 0. Note that we can also use zero as the reset value, because we can calculate always a weight value greater than zero.

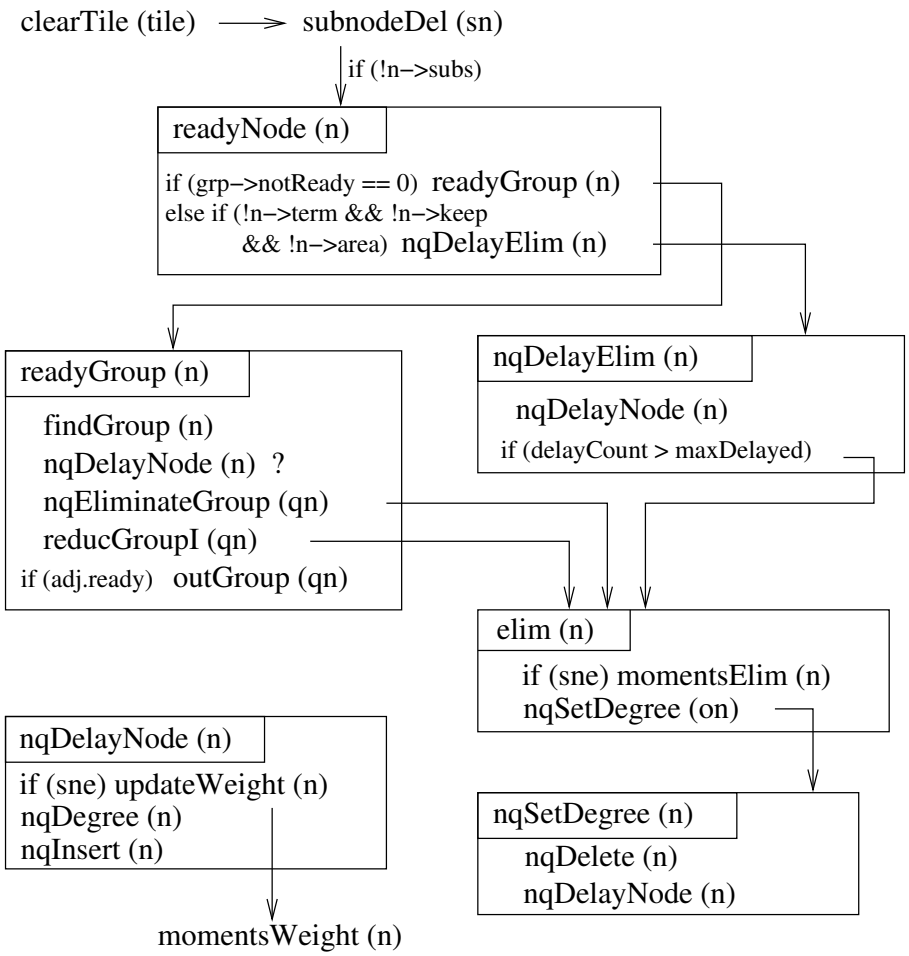
The same strategy can be used for too early eliminated nodes. We take always a node with lowest degree from the queue and first look if a weight needs to be calculated. We can easy calculate the weight, because the node has normally a very low degree. Sometimes, we must take another node, because the node weight is too high.
The weight position (posW) of a node is calculated as follows:




```
if (weight < sneTolerance)
    posW = weight * sneResolution / sneTolerance
else
    posW = sneResolution
```

5. PROCEDURE SCHEME

This scheme gives an overview of the functions called by the *space* program. After function enumTile, function clearTile is called. This can be much later in the process if a band-width is used. Function clearTile shall delete the subnodes of the tile. By resistance extraction, these subnodes are in the node-points. A node can become ready, because of this subnode delete action. A node can be delayed, if no special flags are set. The last node of the group goes directly to function readyGroup. But also readyGroup can delay this last node. This last node cannot give a overflow of the priority queue.



Function findGroup places the nodes of the group in array qn. After function nqEliminateGroup there are not more delayed nodes in array qn. Thus, all delayed nodes are removed from the priority queue and possibly eliminated. Note that the eliminated node is not more in the priority queue.

6. DELAYED NODES

A node group needs delayed nodes in a priority queue, because of making a choice in which order they must be eliminated. A Markowitz scheme is used for this priority queue. But also in case of too many delayed nodes it is useful to eliminate a low priority node of a group. Note that there are only node groups in the res. extraction mode. Thus, delayed[2] nodes have normally always neighbor nodes and resistors. Thus, a weight update or degree calculation is normally always done for a node with resistors. However, there are two situations whereby the delayed node does not have resistors.

Case 1: The node is the only node of the node group, because it is a complete low res conductor and it has also no high res contacts. This node is given to function readyGroup as the last node of the node group. Because the node has no special flags, readyGroup shall delay the node. The node can have many capacitors, is possibly be connected to the ground node or to many other adjacent groups. The degree can be calculated and this degree can be high. And in case of SNE the weight can also be calculated. It is placed (nqInsert) in the priority queue, but is this really needed? After that it must be find back and only in case of SNE there may be a reason not to eliminate this node. And this cost even more time to find it back, because it is in the highest queue at the degree position. Thus, only in case of SNE it might be, that we don't want to eliminate this node. Note that normally in the res. extraction mode this node was an area node, but because of putting off the articulation heuristic it is not special. But normally it is an area node and is not delayed, but it is by reducArtDegree always eliminated because it has no resistors. Note that, when the res. extraction mode is not used, the node is not flagged as area node. In that case there are only single group nodes and these nodes (if not special) are also delayed, but are the only node in the priority queue. And are directly removed from the queue again.

Case 2: The node is the only node of the node group, because all nodes of the group are delayed and all other nodes are already eliminated. This last node is updated by function nqSetDegree and is only not eliminated in case of SNE, if the weight becomes high enough. Without SNE, it is maybe possible to implement a faster eliminate of the total group. With SNE, it can also be possible that no nodes have a weight, because no nodes have a ground capacitance.

---

2. A delayed node is normally not the last node of a node group. Therefor a delayed node has resistors. But also the last node of the group can be delayed by readyGroup itself. And if it is the only node of the group, it does not need to have resistors. It can have substrate resistors to other groups.

7. WEIGHT NOTES

Weight calculation scheme:

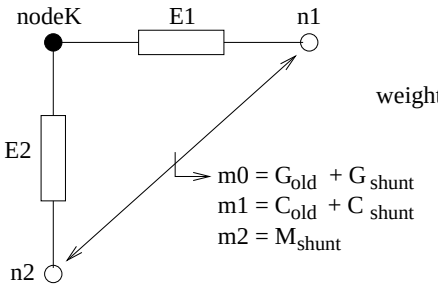
	cx < 0 sumC = 0	cx = 0 sumC != 0
rx < 0 sumG = 0		only CAP
rx >= 0 sumG != 0	only RES sumM != 0 i0 = 0	RES / CAP

In the only RES situation, there are no capacitors connected to the node, there is only a weight when the node has extra moments with a value not equal zero (and i0 = 0).
For sne.fullgraph=2 (all branches) it is also possible that the node has only extra moments2 (to substrate node). This special case is not implemented.

> Only by substrate extraction, when a substrate node may be eliminated (parameter "elim_sub_term_node"), only in that case a substrate capacitance value can be moved over a capacitive branch to another node. In that case only an extra moments2 value can exist. When in that case also the capacitive branch is removed from that node, it is possible that the only RES situation is created. However, because another cap-type is used for substrate caps it is not possible.

Note that in the only RES situation, for the default case (sne.fullgraph=0 (only ground branches) and sne.errorfunc=0), only a weight != 0 for a branch can be calculated when the branch node has a gndCap value != 0. For the other situations, only a gndCap for nodeK (node of weight calculation) is also ok.

Note that only cap-type (cx=0) node capacitances are taken into account. Other cap-types, used for junction and substrate capacitances, are not used. The extra moments of a node can also only be for one cap-type (cx=0).



$$\text{weight}(1,2) = \frac{\text{abs}\ (m2 * O\hat{}2)}{\text{sqrt}\ (m0\hat{}2 + m1\hat{}2 * O\hat{}2)}$$

The shunt values between n1 and n2, are the added values when nodeK is eliminated.
The branch weight(1,2) is the added moment (m2) divided by cap and res parts.
The old values are the existing admittance values between n1 and n2.
The old moment is default (sne.errorfunc=0) not used.

The formules for the shunt values are:

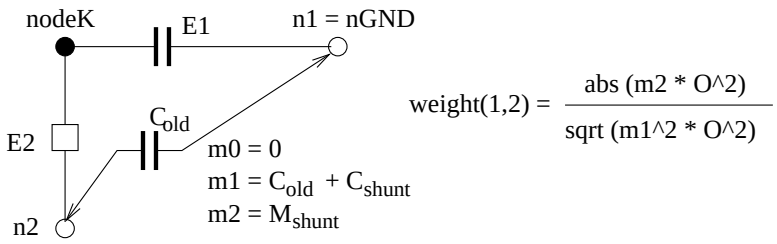
```
Gshunt = (G1*G2) / GS
Cshunt = (C1*G2          + G1*C2 - Gshunt*CS) / GS
Mshunt = (M1*G2 + C1*C2 + G1*M2 - Cshunt*CS - Gshunt*MS) / GS
```

Note that these formules are symmetric, you can interchange n1 and n2. The GS, CS and MS values are the sum-values of respectively all G's, C's and M's connected to nodeK. When we calculate only the weights for the ground branches (default case) and n1 is the ground node, then is G1=0 and in that case the formules are:

```
Cshunt = (C1*G2) / GS                    Gshunt = 0
Mshunt = (M1*G2 + C1*C2 - Cshunt*CS) / GS
```

Note that also Gold=0, thus m0=0, and the formule for the branch weight is:

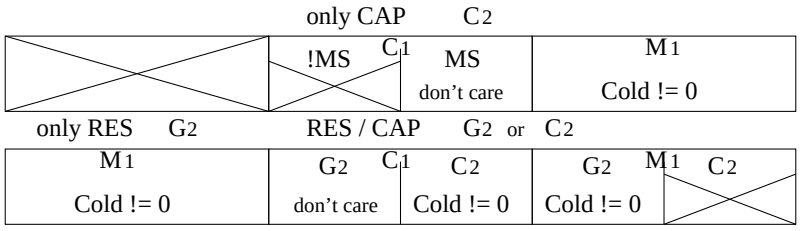
```
weight(1,2) = Omega * abs (m2) / abs (m1)
weight(1,2) = Omega * abs (Mshunt) / abs (Cold + Cshunt)
```



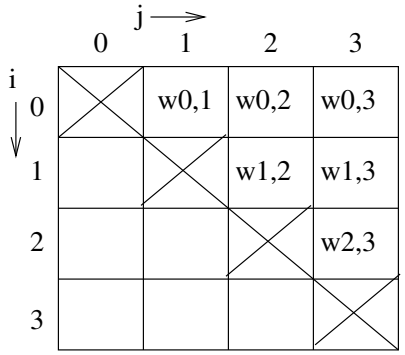
Thus, for ground branches, there is only a branch weight when Mshunt != 0 and Cold+Cshunt != 0. When element E2 is a resistor (G2 != 0 and Cshunt != 0 and CS != 0), this is the case. However, when element E2 is a capacitor (G2=0 and Cshunt=0), then Cold must be != 0. However, when there are only capacitors connected to nodeK, then the formules are (also symmetric):

```
Cshunt = (C1*C2) / CS
Mshunt = (C1*M2 + M1*C2 - Cshunt*MS) / CS
```

In this case Cshunt != 0, but there must be moments (MS != 0), else Mshunt=0.
What happens if C1=0 and M1 != 0? Well, in that case Cshunt=0 and Cold must be != 0. But note, when in a RES environment element E2 is a capacitor (G2=0), then the capacitive branch cannot have a weight (because Mshunt=0). Resumé:



The number of branch weight calculations for nodeK:

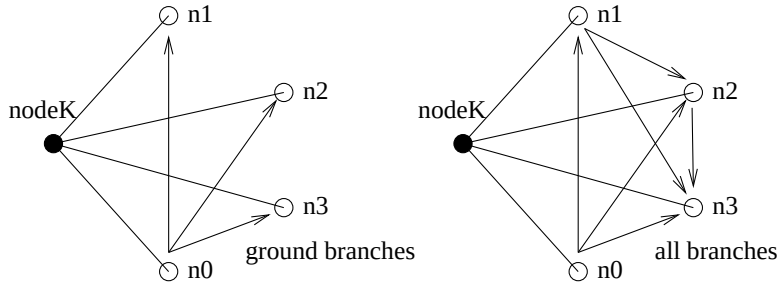


N = 4

ground branches: (N−1)

all branches: N/2 * (N−1)

For the ground branches, only the weights for i=0 are calculated (for N=4 : 3). When the weights for all branches are calculated, then 6 values for N=4. For example nodeK is connected with 3 neighbor nodes and has a ground connection (cap-value or moment-value) to the ground node (n0):



The resulting weight of nodeK is default the square-root of the biggest calculated weight, but below are all possibilities given:

```
sne.norm=0:  weight-nodeK = sqrt( Max(w1,w2,...) )
sne.norm=1:  weight-nodeK = sqrt(w1) + sqrt(w2) + ...
sne.norm=2:  weight-nodeK = sqrt(w1 + w2 + ... )
```

Note:
Ground branches are branches to node GND and not to the substrate node SUBSTR.

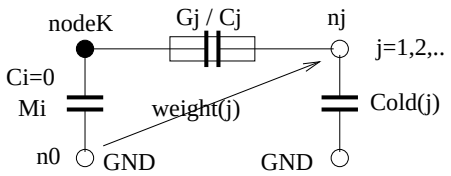
8. DEFAULT WEIGHT CALCULATION

Using the default SNE parameters (sne.fullgraph, sne.errorfunc, sne.norm), it is maybe possible to choice the best branch weight without all calculations. The formules are:

```
=== RES ===                            | === CAP ===
Csh = (Ci*Gj) / GS                     | Csh = (Ci*Cj) / CS
Msh = (Mi*Gj + Ci*Cj - Csh*CS) / GS | Msh = (Ci*Mj + Mi*Cj - Csh*MS) / CS

    weight = Omega * abs (Msh / (Cold + Csh))
```

In the special case Ci=0 (Csh=0) we get the following weight formules (see figure):




```
    RES:   weight = Omega * abs ((Mi*Gj) / (Cold*GS))
    CAP:   weight = Omega * abs ((Mi*Cj) / (Cold*CS))
```

In this case Cold must be != 0 (else no weight). This can also be the only RES situation. For a RES situation the weight is depended of Gj/Cold (undepended of Cj). For a only CAP situation the weight is depended of Cj/Cold (undepended of Mj).

For the general RES situation (Ci != 0) the formules for the weight are:

```
     Gj:   weight = Omega * abs ((Mi/Ci + Cj/Gj - CS/GS) / (Cold/Csh + 1))
    !Gj:   weight = Omega * abs ((Ci*Cj) / (Cold*GS))
```

For a capacitive branch (!Gj) the value of Cold must be != 0 (else no weight). For a resistive branch (Gj) the value of Cold may be 0, in that case the weight is depended of Cj/Gj. Note that, if Cj=0, the weight is undepended of Gj. But in general the weight is depended of:

```
     abs ((X + Cj/Gj) / (Y*Cold/Gj + 1))     # X = Mi/Ci - CS/GS; Y = GS/Ci
```

For the general only CAP situation (Ci != 0) the formules for the weight are:

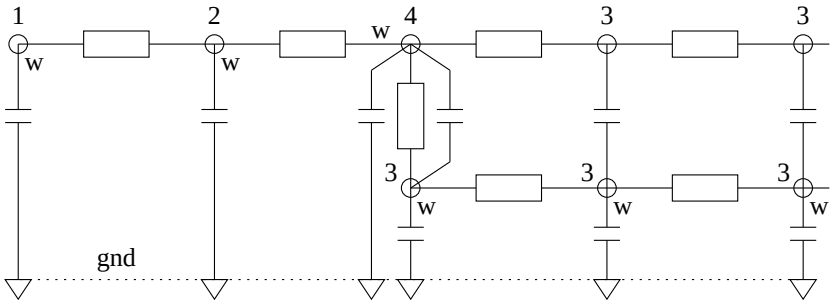
```
     Cj:   weight = Omega * abs ((Mi/Ci + Mj/Cj - MS/CS) / (Cold/Csh + 1))
    !Cj:   weight = Omega * abs ((Ci*Mj) / (Cold*CS))
```

In general the value of Cold may be 0, in that case the weight is depended of Mj/Cj. Note that in that case, if Mj=0, the weight is undepended of Cj. In case Cj=0 there must be a Mj and Cold must be != 0 (else no weight). But in general the weight is depended of:

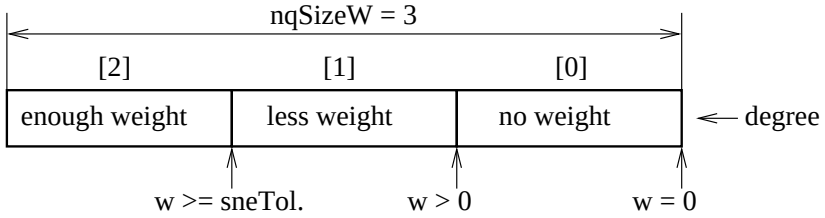
```
     abs ((X + Mj/Cj) / (Y*Cold/Cj + 1))     # X = Mi/Ci - MS/CS; Y = CS/Ci
```

9. NODE PRIORITY

In a node group, there can be nodes with a weight and nodes without a weight. When the weight values are lower than sneTolerance there is no difference between nodes with and without a weight (see figure).

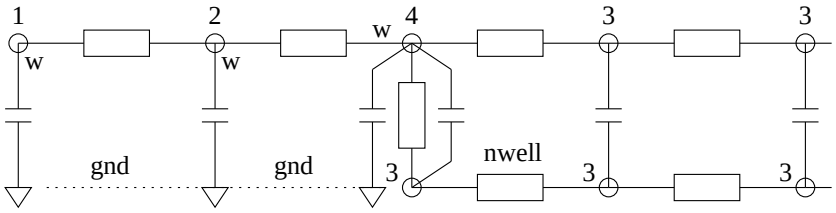


Thus it is possible that nodes with a weight are first eliminated and that makes nodes without a weight possibly more important (giving it a weight). Note that a ground capacitor is also not counted for the degree. Therefor it is maybe better to add an extra priority queue position for a weight > 0 as follows:

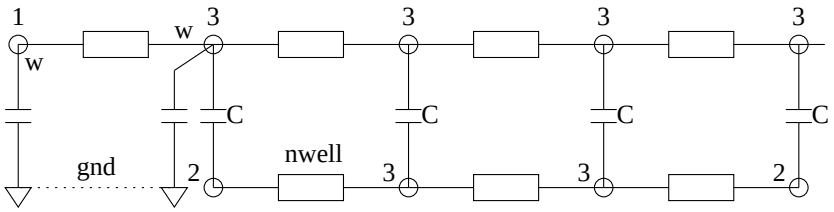


Thus we can first eliminate the nodes without a weight.

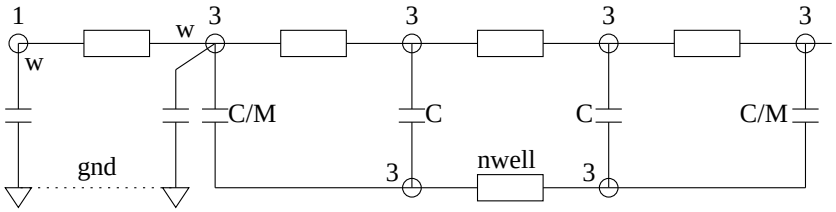
This happens also by a nwell area:



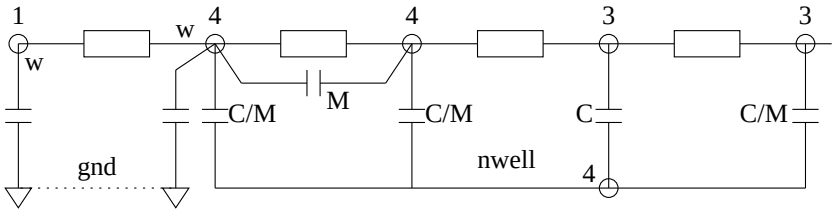
What, if the nwell area is not connected with a contact to interconnect?



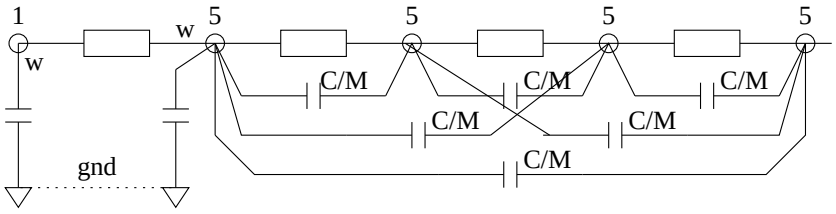
After elimination of two nwell nodes:



After elimination of for last nwell node:



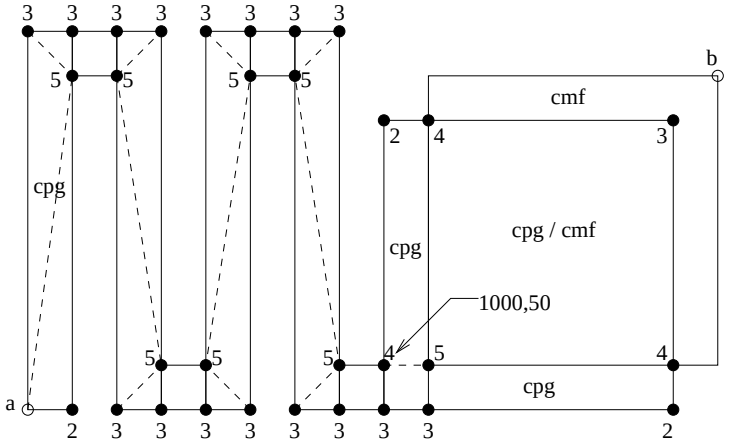
The resistor gets a moment, this is a capacitor element without cap value. The nwell group contains now only one node, with only capacitors connected. It is maybe not a good idea to eliminate this last nwell node. After elimination we get:



If we don't eliminate the nwell node, there can be many moments parallel to the interconnect resistors. Note that if we make the nwell area low res, then we don't have these moments. However, if we eliminate the nwell node, there can be many capacitors parallel to the interconnect resistors.

10. RC EXAMPLE

In the figure below you see the layout of a simple RC design example (cell RCshort).



The R-part is made with the poly-silicon mask (cpg) and has one terminal point "a". The C-part is between mask cpg and the first metal mask (cmf). The cmf mask is low ohmic and has one terminal "b". Thus the metal group has only one node and the poly group has 32 nodes. By a normal resistance extraction (option -r) and not using parameter "equi_line_ratio" the above triangular resistance mesh is used and there are 31 delayed nodes in the poly group (the numbers by the nodes are the default initial degree).

> Note that each poly node has a ground capacitor, this couple cap is not counted for the node degree. Four poly nodes have a couple cap with the metal conductor, this cap is counted for the node degree.

The following circuit is extracted after the elimination of the delayed nodes:

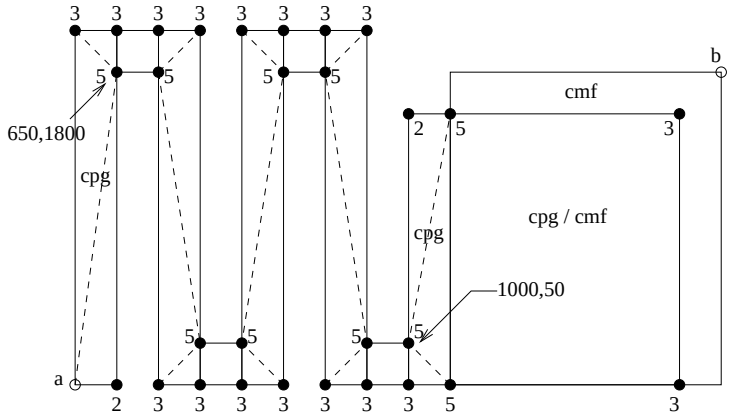
```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=1e6 -rCG RCshort
% xsls RCshort

network RCshort (terminal b, a)
{
    cap 3.96725f (b, a);
    cap 2.498938f (b, GND);
    cap 14.12513f (a, GND);
}
```

When we chose a higher frequency, first one of the delayed nodes shall get enough weight to be kept in the network. This node depends of the order of elimination. First the lowest degree nodes are eliminated (degree of 2). On the end the node on coordinate position 1000,50 stays in the network. That looks a good choice, because in that case we get a ‘‘good’’ resistor value. See the extracted circuit on next page (node 1 is the node on coordinate position 1000,50). However, when we change the layout a little (cell RCshort2), we get another node that is kept (on position 650,1800).

```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=1e7 -rCG RCshort
% xsls RCshort

network RCshort (terminal b, a)
{
    cap 319.5513e-21 (b, a);
    cap 3.96693f (b, 1);
    cap 2.498938f (b, GND);
    res 5.682549k (a, 1);
    cap -1.544651f (a, 1);
    cap 4.873224f (a, GND);
    cap 9.251901f (1, GND);
}
```

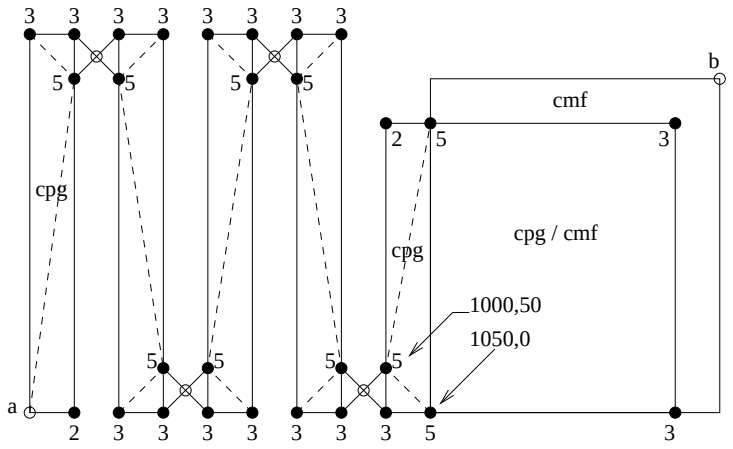



```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=6e7 -rCG RCshort2
% xsls RCshort2

network RCshort2 (terminal b, a)
{
    cap -10.43673e-18 (b, a);
    cap 3.739812f (b, 1);
    cap 2.937313f (b, GND);
    res 1.408717k (a, 1);
    cap 59.20282e-18 (a, 1);
    cap 1.148447f (a, GND);
    cap 12.95068f (1, GND);
}
```

Node 1 is the node on coordinate position 650,1800. We see, that it is not a good choice, because we get a ‘‘small’’ resistor value. Note that also the capacitance value between node 1 and "b" looks too small. We also must use a little bit higher frequency to keep one node.

What happens if we use parameter "equi_line_ratio=1.0" for this layout? See the figure on the next page, we get 4 extra line nodes which are not delayed. These line nodes separate the delayed nodes from each other and therefor they get smaller weights.



Because the line nodes separate the delayed nodes, a much higher frequency is needed to keep one of the delayed nodes. See the extraction and the resulting network below:

```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=1e10 -rCG RCshort2
% xsls RCshort2

network RCshort2 (terminal b, a)
{
    cap 998.8494e-21 (b, a);
    cap 3.728376f (b, 1);
    cap 2.937313f (b, GND);
    res 5.685341k (a, 1);
    cap -1.547329f (a, 1);
    cap 4.876754f (a, GND);
    cap 9.222371f (1, GND);
}
```

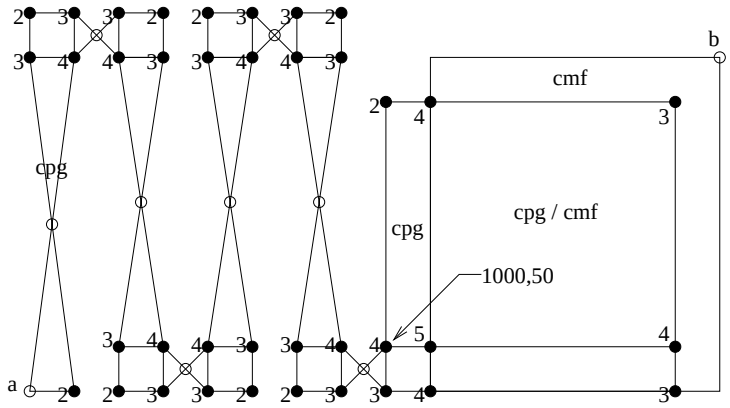
Node 1 is the node on coordinate position 1050,0 and is a good choice. Note that the line nodes are eliminated in the art. reduction step, because the resulting "res_cnt" on these nodes is lower than "min_degree". Note that no weight is calculated for the line nodes, because they are not delayed. For the other layout RCshort we get:

```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=1e9 -rCG RCshort
% xsls RCshort

network RCshort (terminal b, a)
{
    cap 375.8627e-21 (b, a);
    cap 3.966874f (b, 1);
    cap 2.498938f (b, GND);
    res 5.68188k (a, 1);
    cap -1.54472f (a, 1);
    cap 4.872886f (a, GND);
    cap 9.252239f (1, GND);
}
```

Node 1 is the node on coordinate position 1000,50 and is a ‘‘good’’ choice.

By a resistance extraction with option -z (mesh refinement) and also using parameter "equi_line_ratio", we get another mesh and more line nodes:



The extraction result for RCshort2 is (node 1 is 1000,50):

```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=1e10 -rCG RCshort2
% xsls RCshort2

network RCshort2 (terminal b, a)
{
    cap 348.8009e-21 (b, a);
    cap 3.729026f (b, 1);
    cap 2.937313f (b, GND);
    res 5.848502k (a, 1);
    cap -1.547296f (a, 1);
    cap 4.886612f (a, GND);
    cap 9.212513f (1, GND);
}
```

The extraction result for RCshort is (node 1 is 1000,50):

```
% space -E space.MTB2.t -P space.MTB.p -Ssne.frequency=1e9 -rCG RCshort
% xsls RCshort

network RCshort (terminal b, a)
{
    cap 365.1545e-21 (b, a);
    cap 3.966885f (b, 1);
    cap 2.498938f (b, GND);
    res 5.848502k (a, 1);
    cap -1.547314f (a, 1);
    cap 4.886614f (a, GND);
    cap 9.238511f (1, GND);
}
```

Note that again RCshort2 has less couple capacitance between node 1 and node "b".
We must conclude that the use of line nodes have a good effect on SNE extraction.