Layout Synthesis Methods for Integrated Circuits

Compendium

H. Graeb

Version 1.2 (WS 16/17) Helmut Graeb

The presentation follows in many parts: “Kurt Antreich: Computer-Aided Layout Design, Lecture Notes, Technische Universität München, ~1980 – 2003”.

There is no textbook that corresponds to this compendium. A textbook which is still pretty close is:

- Sabih H. Gerez, Algorithms for VLSI Design Automation, John Wiley & Sons, 1999

Other textbooks for layout synthesis are the following. Just check which one fits your learning style best:

- Sadiq Sait, Habib Youssef, VLSI Physical Design Automation – Theory and Practice, McGraw-Hill, 1995
- Sung Kyu Lim, Practical Problems in VLSI Physical Design Automation, Springer, 2008
- Naveed Sherwani, Algorithms for VLSI Physical Design Automation, Kluwer, 2004
- Thomas Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley-Teubner, 1990
- Andrew Kahng, Jens Lienig, Igor Markov, Jin Hu, VLSI Physical Design: From Graph Partitioning to Timing Closure, Springer, 2011

Status: February 2, 2017

Layout Synthesis Methods for Integrated Circuits
Compendium
H. Graeb
Technische Universitaet Muenchen
Institute for Electronic Design Automation
Arcisstr. 21
80333 Munich, Germany
graeb@tum.de
Phone +49.89.289.23679

Content

1. Introduction

1.1 Design views, transistor netlist, stick diagram

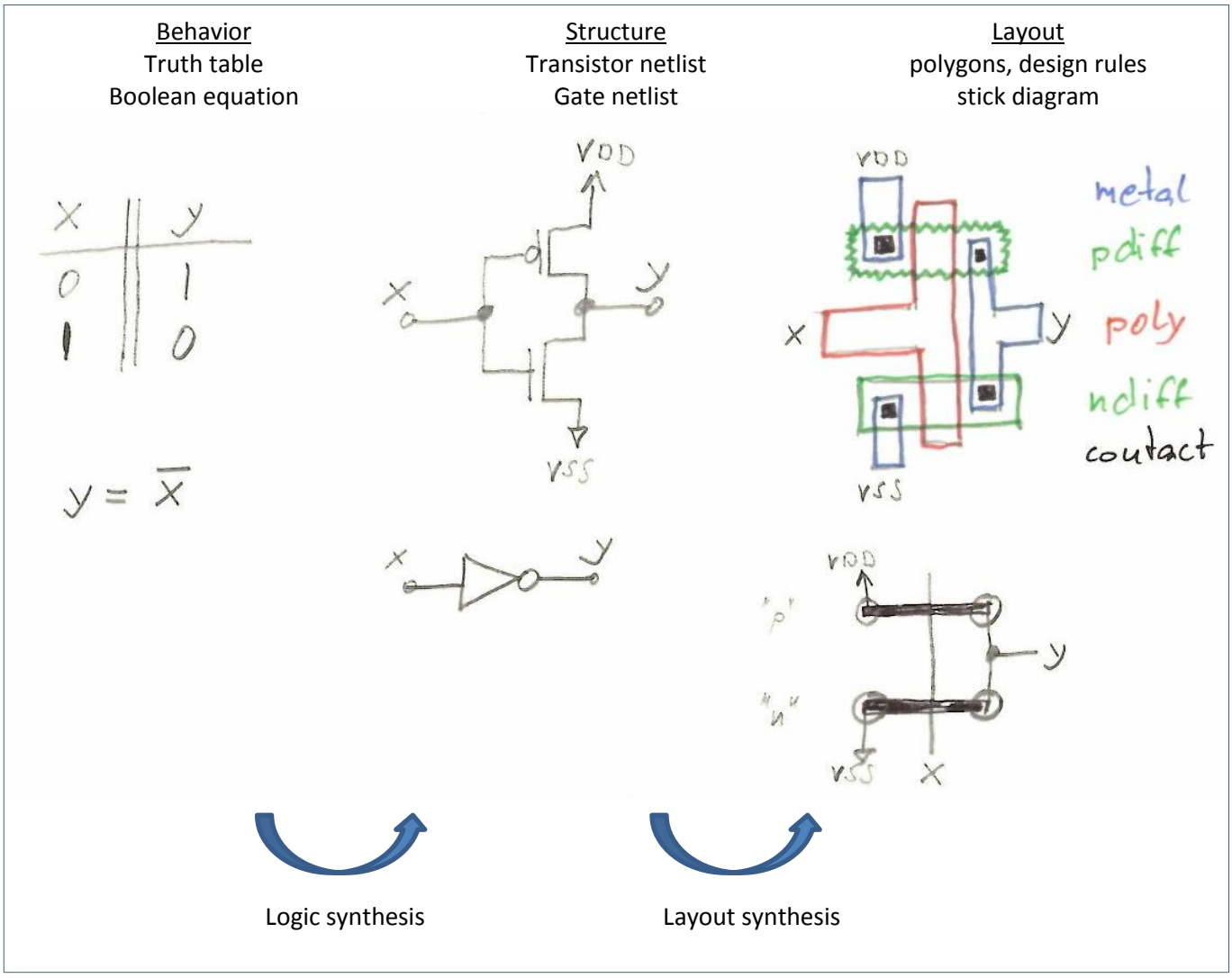
Synthesis of electronic circuits includes two dimensions:

1. An accumulation of implementation details happens ("**refinement**").
2. A transition of the design view happens. This part is usually considered to be the real "**synthesis**" process, rather than the refinement.

- The three basic **design views** are circuit/system **behavior**, **structure** and **layout**.
- The transition from the behavioral view to the structural view is called **logic synthesis**.
- The transition from the structural view to the layout view is called **layout synthesis**.

This systematic should be considered as simplified and fuzzy. An industrial design flow is by far more complex! The following figure illustrates the different design views for a simple CMOS inverter.

Figure 1: Design views of a CMOS inverter



The process of enrichment is illustrated in Figure 1 by different descriptions on the structural level and the layout level, which are more or less abstract. For the inverter structure, for instance, either a gate symbol or the netlist of the transistors, which contains more implementation details that the simple gate symbol, can be given. The layout, for instance, can be given in form of polygons that could already comply with the design rules, which refer to complex distance constraints between polygons. Alternatively, a stick diagram gives a coarse layout plan as a preliminary step in the layout design process. Please note that the source/drain direction in Figure 1 is horizontally, but could also done vertically.

Corresponding design views can be created for other digital cells like a NAND gate (Figure 2) or a NOR gate (Figure 3).

Figure 2: Design views of a NAND gate

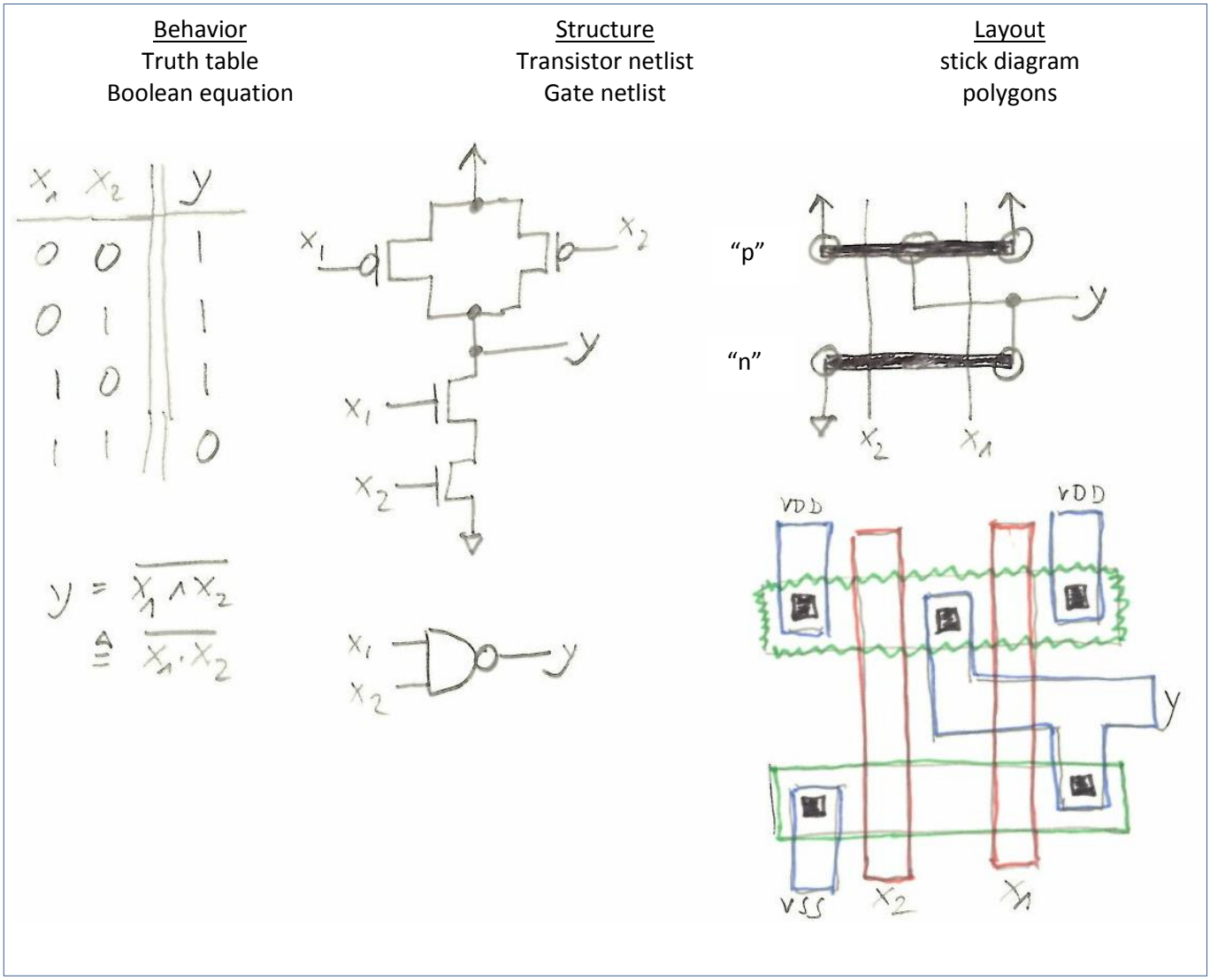
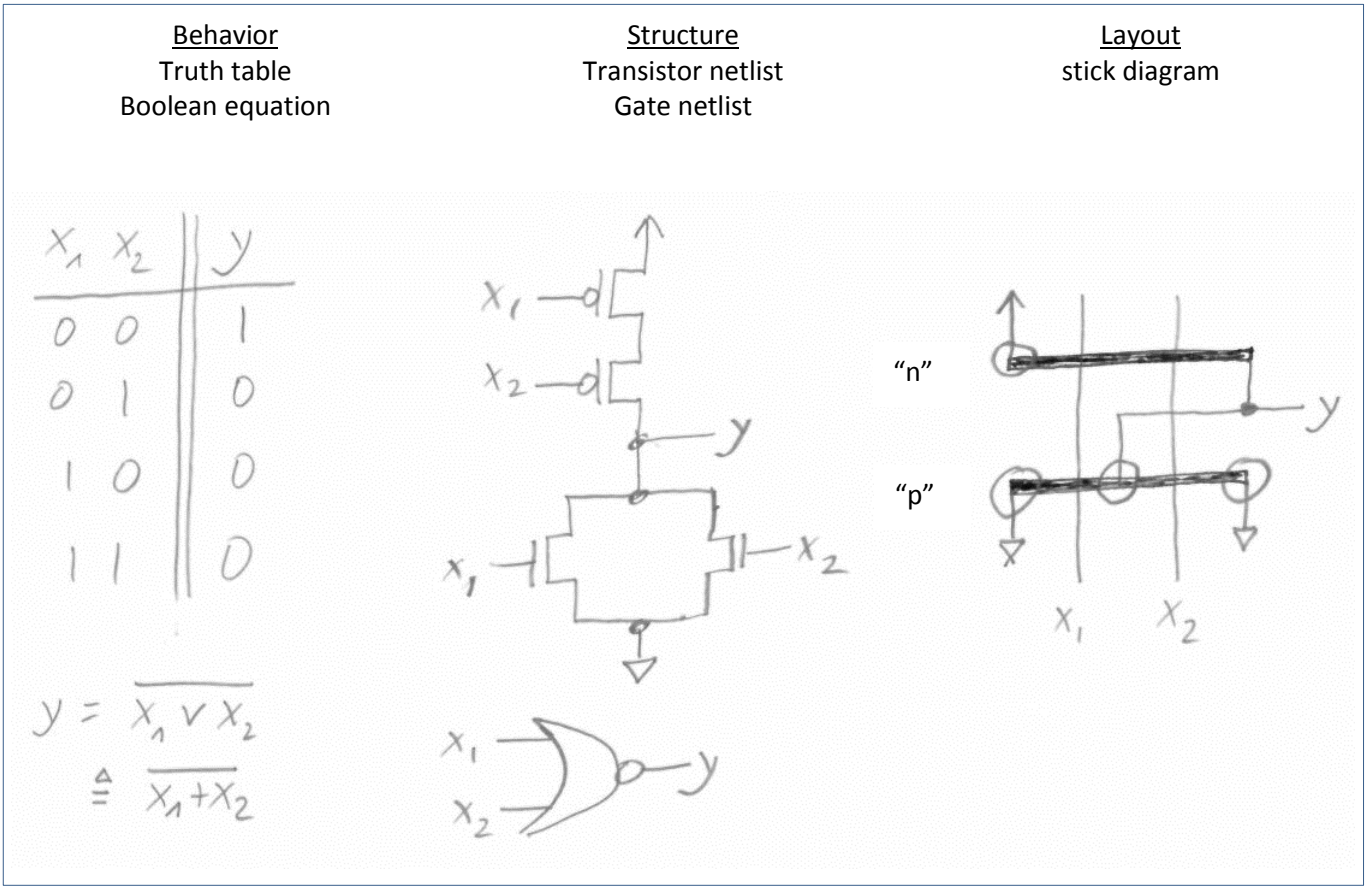


Figure 3: Design views of a NOR gate



Looking at the transistor netlists of the NAND and NOR gates, we can see that in CMOS logic each input goes to both a PMOS transistor and to an NMOS transistor. If two inputs are "or-ed" the corresponding PMOS transistors are put in series and the corresponding NMOS transistors are put in parallel. If two inputs are "and-ed" the corresponding PMOS transistors are put in parallel and the corresponding NMOS transistors are put in series. The output signal is at the node where the PMOS and the NMOS parts are connected, and it is the inverted signal of the "or-ed" and "and-ed" inputs.

This property between logic function and transistor netlist is also illustrated in the following example (Figure 4), where the parallel and series connections resulting from "or-ing" are marked in green and from "and-ing" in blue. It is also indicated how complex logic functions result in a sequence of hierarchical parallelizing and serializing of blocks according to the order of logic operations. In this example, the "or-ing" (green) happens on top of the "and-ing" (blue).

Figure 4: Transistor netlist of logic function $y = \overline{x_1 \cdot x_2 + x_3 \cdot x_4}$

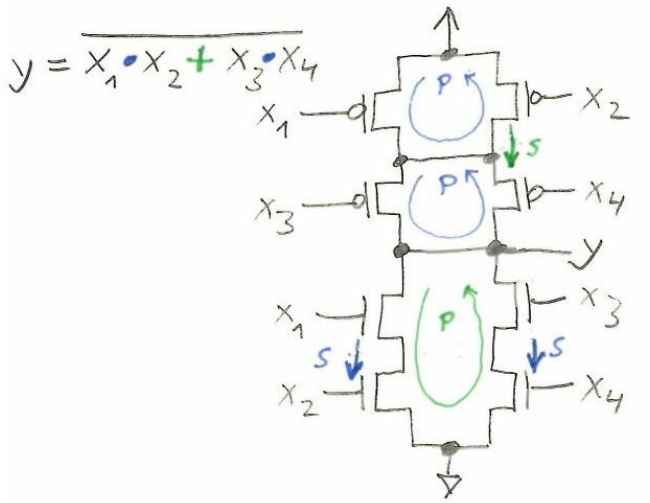
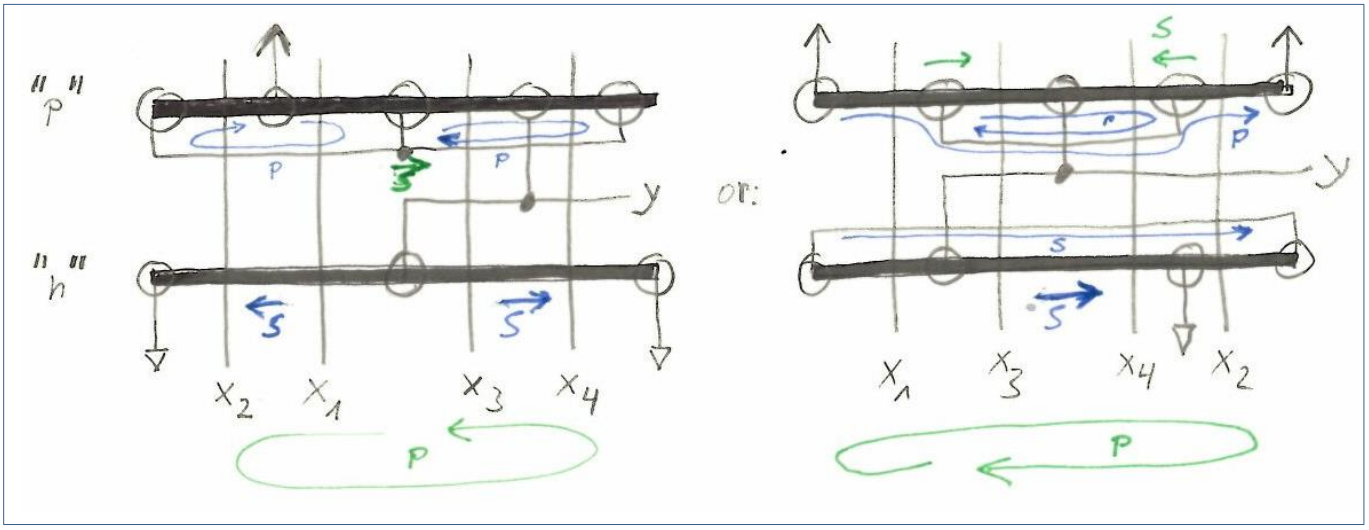


Figure 5 shows two alternative stick diagrams of the logic function in Figure 4. The parallel/series connections that correspond to the logic function and the transistor netlist have been marked accordingly. The left layout appears to be the better one, as we have a smaller number of contacts, shorter/less routing, and shorter signal paths. For the stick diagram on the left side, it was utilized that for the NMOS transistors there are two pairs of two transistors in series, which can share a source/drain contact. This has been the starting point of the stick diagram on the left side. Hence we look for a method to arrange the order of input signals such that a maximum number of shared source/drain contacts can be achieved. Such a method will be presented in the following.

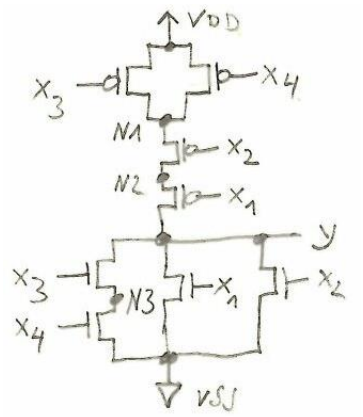
Figure 5: Two alternative stick diagrams of logic function $y = \overline{x_1 \cdot x_2 + x_3 \cdot x_4}$



1.2 Stick layout design by Euler paths in the structural graph

The method to design a stick layout will be illustrated with the following example:

Figure 6: Transistor netlist of logic function $y = \overline{x_1 + x_2 + x_3 \cdot x_4}$

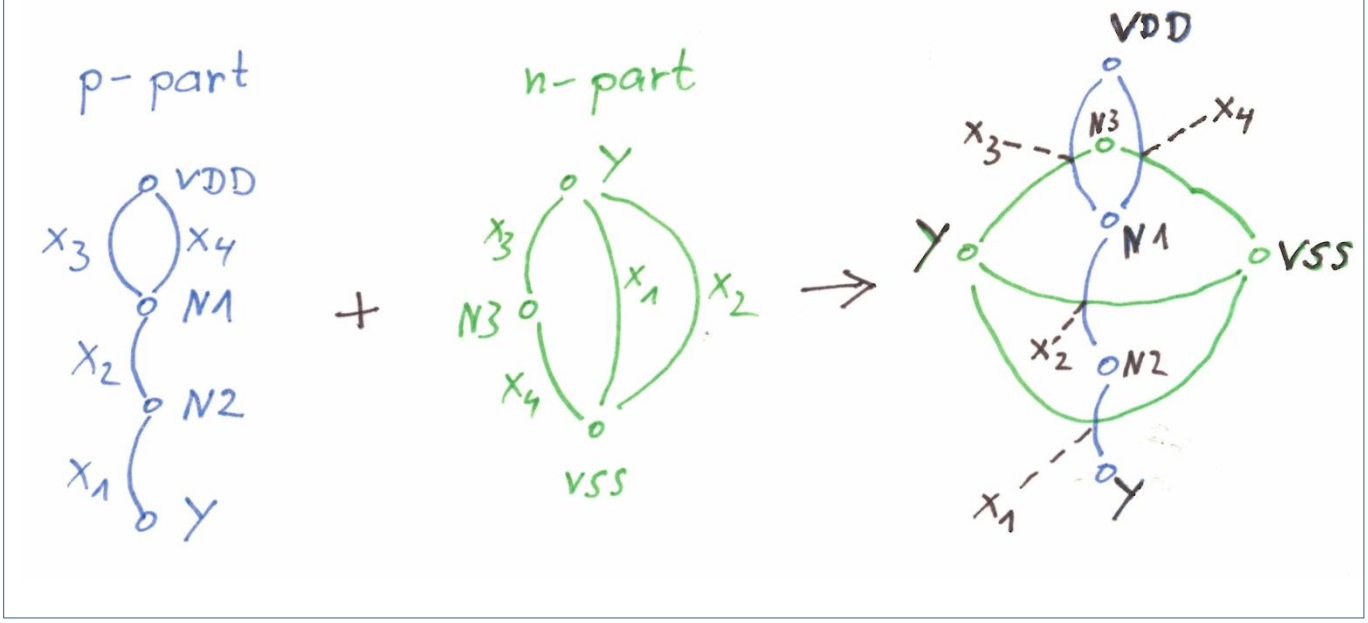


A structural graph that represents the transistor netlist and logic function can be set up. The following correspondences between netlist and graph hold:

- Vertex = netlist node/signal
- Edge = netlist transistor
- Edge label = input/gate signal

A p-part for the PMOS transistors between VDD and the output node, and an n-part for the NMOS transistors between the output node and VSS can be set up. This is illustrated in Figure 7 for the example of Figure 6.

Figure 7: Structural graph of logic function $y = \overline{x_1 + x_2 + x_3 \cdot x_4}$



As one input signal always refers to the gate of one PMOS transistor and to the gate of one NMOS transistor at a time, the number of edges of n- and p-part of the structural graph is equal, and there is always one pair of edges with the same edge label of the respective gate signal.

Therefore, an overall structural graph can be drawn by putting n-part and p-part on top of each other in such a way that the two edges of n- and p-part that correspond to one input are crossing each other. This is also illustrated in Figure 7.

- An **Euler path** is a **path over all edges** in a graph such that **each edge is passed exactly once** in this path.
- If the **start vertex and end vertex are identical**, an Euler path is denoted as **Euler circle***/tour*.

An Euler path in the structural graph therefore represents an ordering of transistors such that source/drains of the transistors may be shared (connected, abutted).

If the sequence of edge labels (i.e., gate signals) is the same for an Euler path in the n-part and an Euler path in the p-part, then the gate signal is only needed in one column of the layout stick diagram. This leads to the following solution approach.

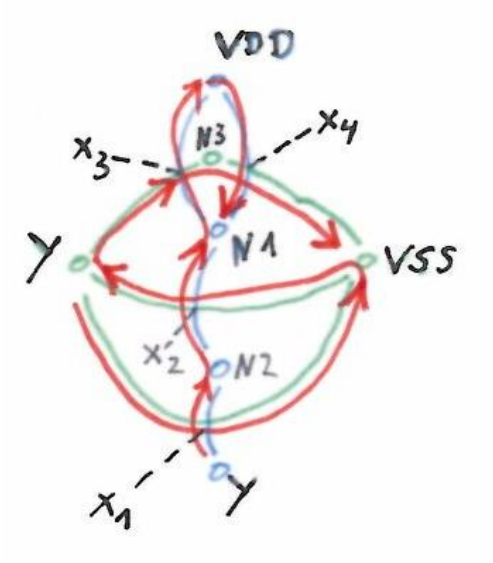
Solution approach

1. Find all Euler paths of n-part and of p-part of the structural graph.
2. Find an n-part Euler path and a p-part Euler path with identical labeling.
3. If 2. is not possible then break gates to achieve 2. while minimizing the number of breaks.

In the example, we can find Euler paths with identical labeling in the n- and p-part according to 2. (see Figure 8):

$$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_3$$

Figure 8: Euler paths in n- and p-part of Figure 7 with identical labeling

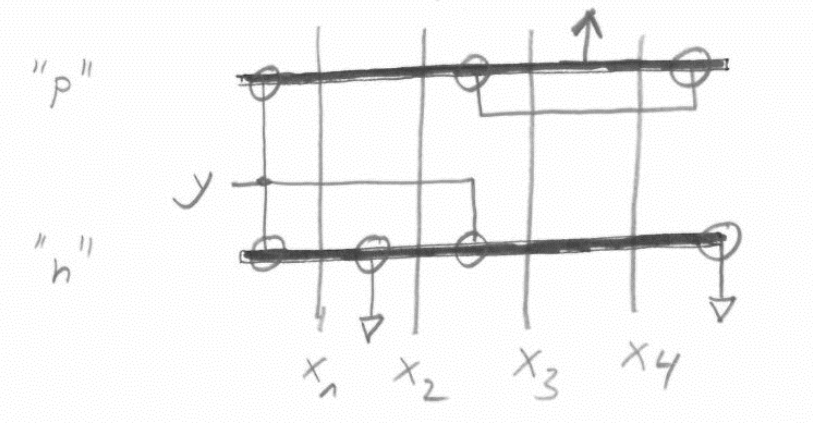


The Euler paths in the example refer to these signals in the n-part: $y \rightarrow v_{ss} \rightarrow y \rightarrow n_3 \rightarrow v_{ss}$

The Euler paths in the example refer to these signals in the p-part: $y \rightarrow n_2 \rightarrow n_1 \rightarrow v_{dd} \rightarrow n_1$

The stick diagram can immediately be given from the Euler paths and corresponding signal orders:

Figure 9: Stick diagram of the logic function of Figure 6



Euler paths can be computed with the Fleury algorithm, which is of the order $\mathcal{O}\ (|E|^2)$, where E is the set of edges.

Euler circles can be computed with the Hierholzer algorithm, which is of the order $\mathcal{O}\ (|E|)$.

Euler paths exist if there are 0 or 2 vertices with odd degree (i.e., nr. of adjacent edges).

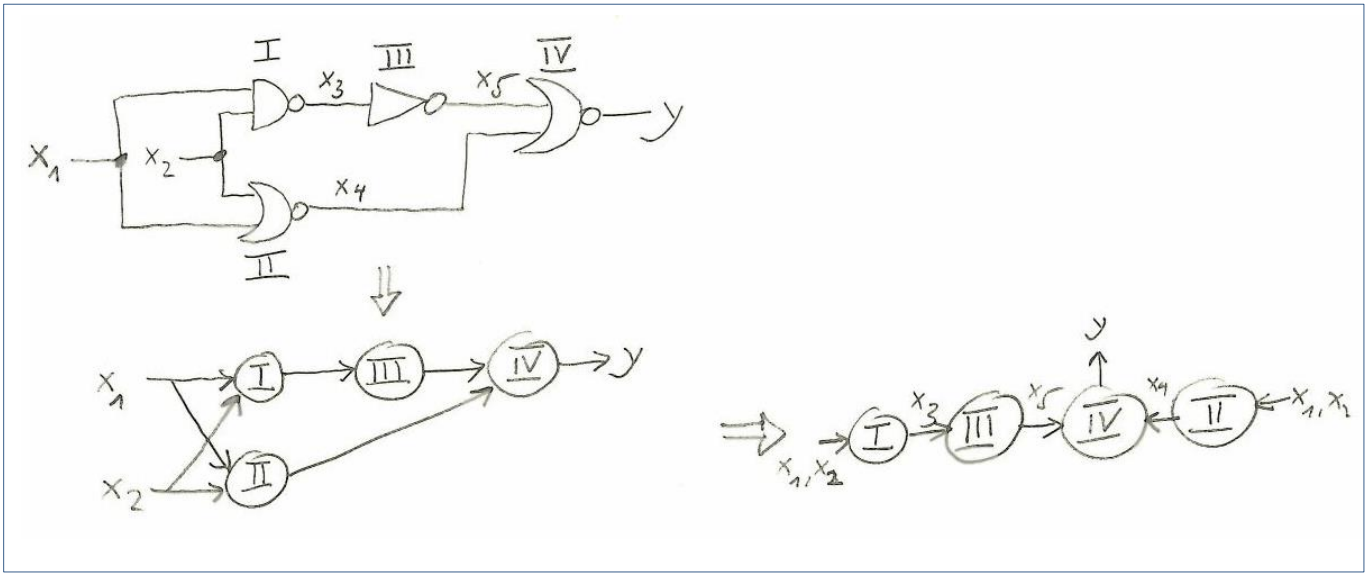
Euler tours exist if all nodes have even degree.

Euler tours exist if all edges can be obtained by the union of pairwise disjunct “circle paths” (starting point for Hierholzer algorithm).

1.3 Hierarchical and stacked stick layout design

Usually, (digital) circuit netlists are given as hierarchical gate netlists, where gates are elements of an often quite complex cell library. Figure 10 shows an illustrative example with 4 gates. From the layout point of view, these gates become modules, which are first placed (i.e., their position on the chip is determined) and then routed (i.e., the connections according to the netlist are laid out). This represents a hierarchical proceeding, where the layout of the modules is predetermined and used within the layout of the whole circuit. In Figure 10, a row-like placement of the four modules is determined easily. For complex circuits, the placement problem refers to millions of modules.

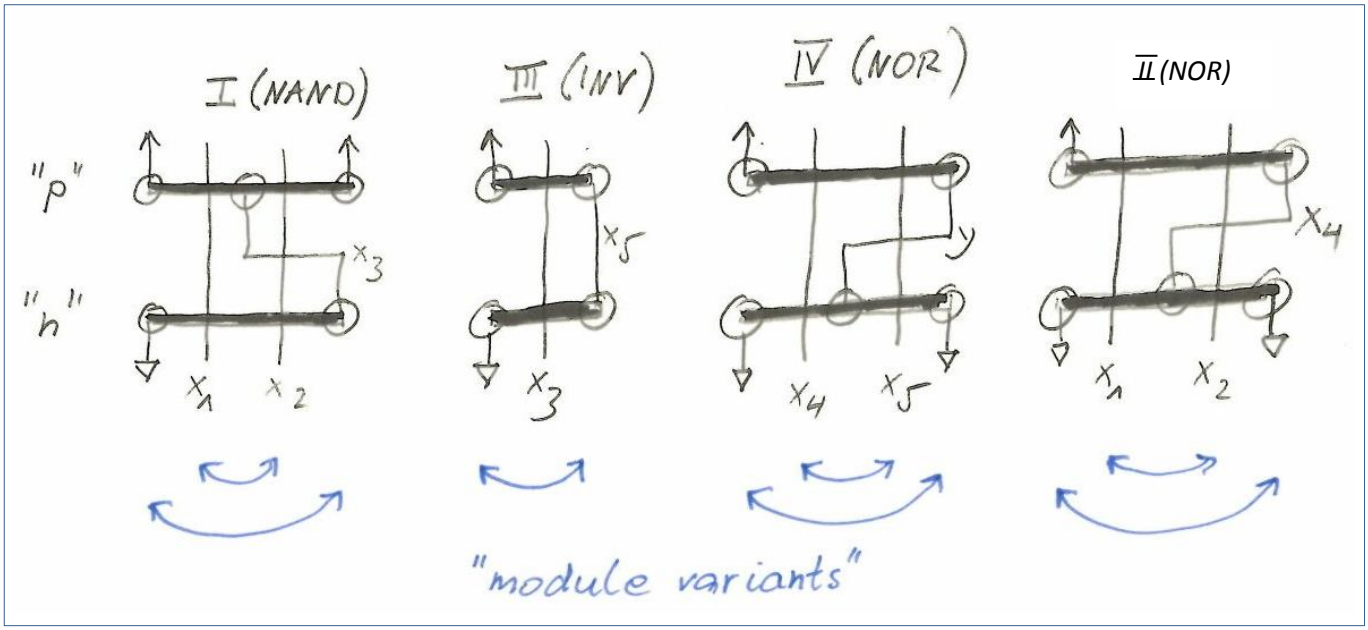
Figure 10: Gate netlist example and corresponding placement of gates (modules)



Aiming at a row-like stick diagram of the four modules according to Figure 10, the stick diagrams of the four modules (NAND, INV, 2 x NOR) are used. These are given in Figure 11. For the row arrangement with one p-diffusion row on top of one n-diffusion row, there are several variants for each module, which can be obtained by mirroring the whole cell or exchanging the columns of the input signals. These variants are indicated in blue in Figure 11.

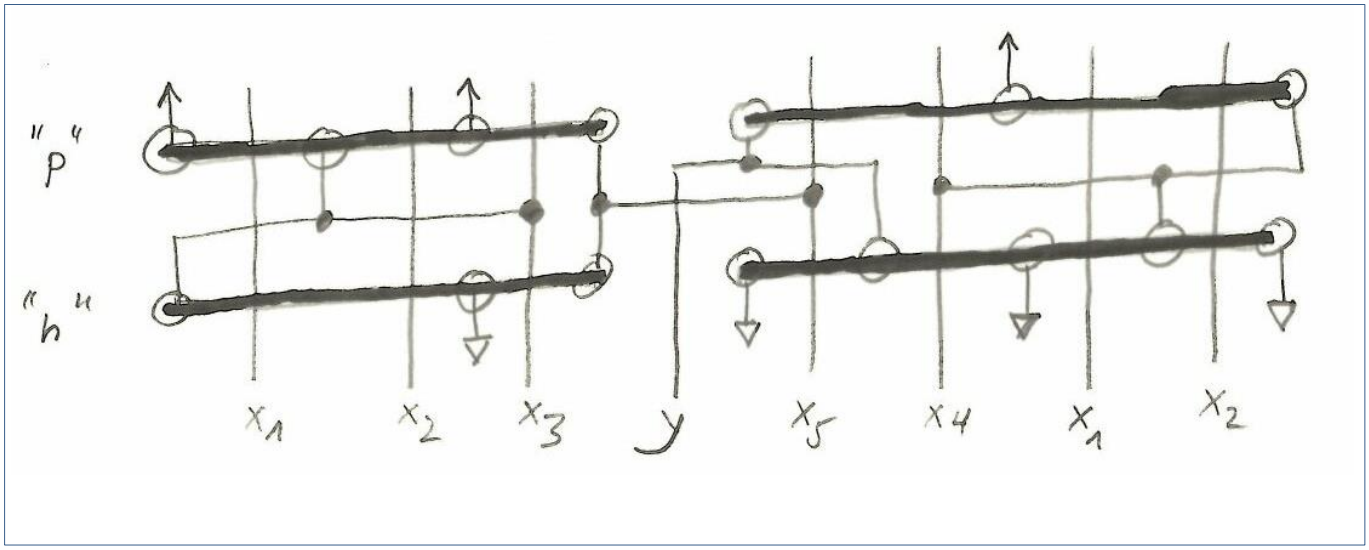
More complex types of variants may exist in general. Computing such variants is often called **module generation** and is of great importance for basic building blocks of analog circuits.

Figure 11: Stick diagrams of modules of circuit in Figure 10 with indicated module variants



A corresponding layout for the whole circuit of Figure 10 is given in Figure 12. We can see that module I (NAND) has been mirrored along its center vertical axis to merge the connections to VDD and VSS with module III (INV), and that module IV (NOR) has been mirrored the same way for the same reason and in order to route signal x_5 to module III and signal x_4 to module II on a shorter distance.

Figure 12: Stick diagram of circuit in Figure 10



We can see that the complexity of the layout problem is growing quickly with the size of the circuit.

Alternative ways have been developed, like

- Stack transistors on gate signal (multi-diffusion rows, stacked stick diagram)
- Gate-matrix layout
- Symbolic layout.

The book “Weste/Eshraghian: Principles of CMOS VLSI Design – A Systems Perspective, Addison-Wesley 1985” gives more details on the “old days” of CAD-supported manual layout design.

Modern layout synthesis tools are based on optimization methods for the placement of modules and subsequent routing. Some basics for these methods are treated in the rest of this compendium.

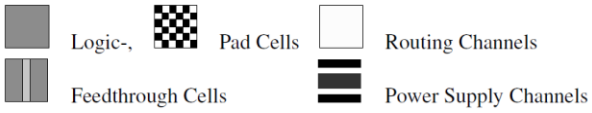
1.4 Application-specific integrated circuit (ASIC) design styles

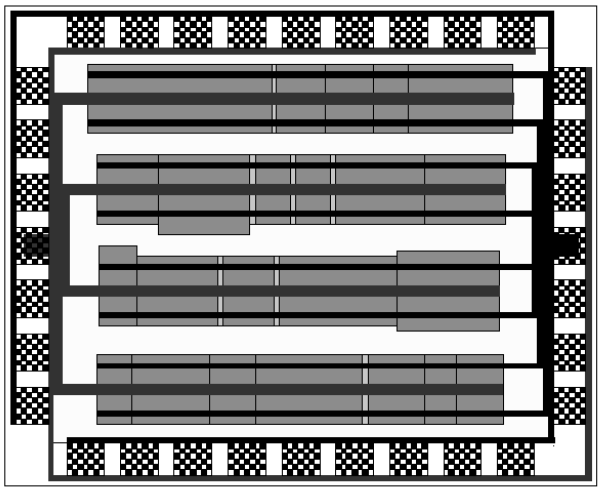
There are two basic design styles for ASICs, standard cells and macro cells. In complex system-on-chips (SoCs) they appear in combination. Figure 13 outlines the corresponding layout schemes.

Standard cells are arranged in rows, where the height is hardly varying, and where the cell size is varying over the cell width (see: stick diagram). The power routes (VDD, VSS) are integrated into the rows. The routing is arranged in so-called channels that alternate with the cell rows. The channels can have variable width according to the required horizontal routing tracks within the channel. The cells pins are arranged column-like. Vertical routing can happen at the left or right end of the rows or along vertical feed through cells in a cell row.

Macro cells have very different sizes and aspect ratios, a preliminary placement is obtained by so-called floorplanning. The placement within a macro-cell can be a standard cell design or another macro cell design. Placement and routing are more complex than in standard cell design.

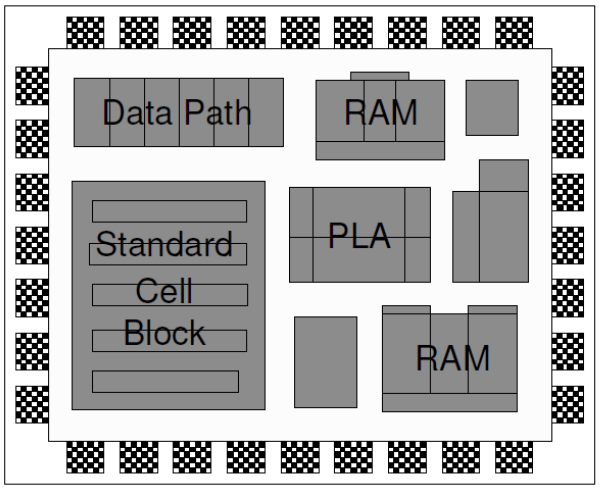
Figure 13: Standard cell layout and macro cell layout





Standard Cells

cell height nearly unique
variable cell width
channel routing
variable channel width



Macro Cells

variable cell size
floorplanning
channel routing
switchbox-routing, L-channels
design of power supply

2. Circuit topology description

2.1 Netlist description

The netlist describes the connectivity of the circuit and consists of three elements, which are *modules*, *pins*, (signal) *nets*:

Eq. 1 **Module** (index) **set**: $M = \{\mu | \mu = 1, \dots, n_M\}$

Eq. 2 **Pin** (index) **set**: $P = \{\rho | \rho = 1, \dots, n_P\}$

Eq. 3 **Net** (index) **set** $N = \{\nu | \nu = 1, \dots, n_N\}$

The *netlist* consists of two relations, the *module-pin relation* and the *pin-net relation*:

Eq. 4 **Module-pin relation**: $MP = \{(\mu, \rho) | \mu MP \rho\} \subseteq M \times P$

Eq. 5 **Pin-net relation**: $PN = \{(\rho, \nu) | \rho PN \nu\} \subseteq P \times N$

The pin-net relation for instance reads that PN is a circuit-specific set of pairs of pin and net, which is a subset of all possible pin-net pairs, i.e., the cross product of the sets of pins and nets.

The *simplified netlist* skips the pins and directly establishes the *module-net relation*:

Eq. 6 **Module-net relation**: $MN = MP \circ PN = \{(\mu, \nu) | \mu MN \nu\} \subseteq M \times N$

The module-net relation results from concatenation of the module-pin relation and the pin-net relation: $MN = MP \circ PN$. It holds that $(\mu, \nu) \in MN \Leftrightarrow \mu MN \nu \Leftrightarrow \mu (MP \circ PN) \nu \Leftrightarrow \exists_{\rho \in P} [\mu MP \rho \wedge \rho PN \nu]$

The **circuit netlist** is mathematically a *graph* with vertice set V and edge set E:

Eq. 7 $G_{circ} = (V_{circ}, E_{circ}) = (M + P + N, MP + PN),$

where the vertices V are the modules, pins and nets, and where the edges E are the module-pin and pin-net relations. Accordingly, the **simplified netlist** is

Eq. 8 $G'_{circ} = (V'_{circ}, E'_{circ}) = (M + N, MN)$

Figure 14 illustrates the circuit graph for a circuit example. On the left side, the module-pin relation and the pin-net relation are given. On the right side, the module-net relation that results from the concatenation of these two relations is given. Please note that each of these graphs is a bipartite graph consisting of two disjunct sets, where the elements of each set are not connected among each other.

Figure 15 gives an idea of a circuit topology of the example netlist in Figure 14. Please note that in Figure 15, more details have been included compared to the netlist. There is a first idea on the placement of modules and on the routing of signal nets, and the sizes of the modules and the locations of pins in each module have been considered (module-pin-net relations as in Figure 14 do not include the module size and pin location).

Figure 14: Example of netlist

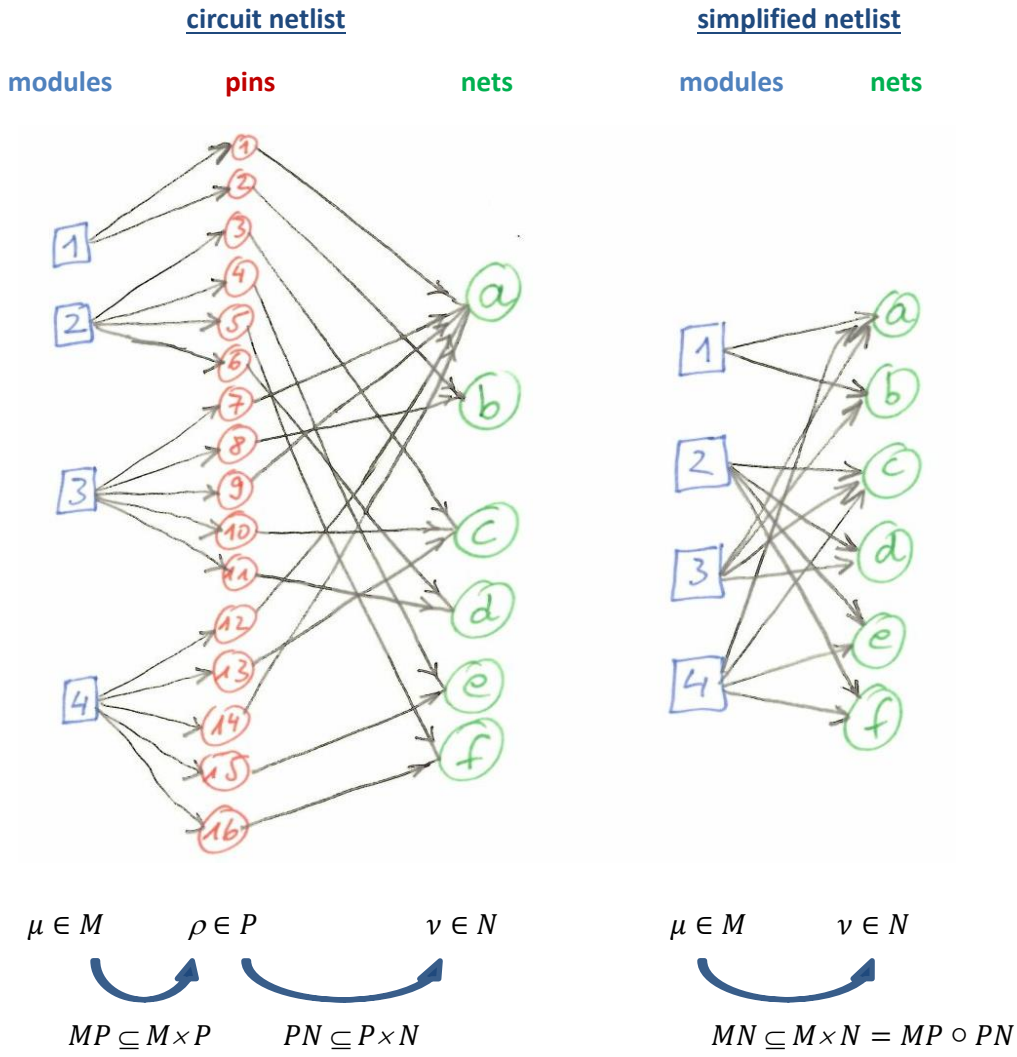
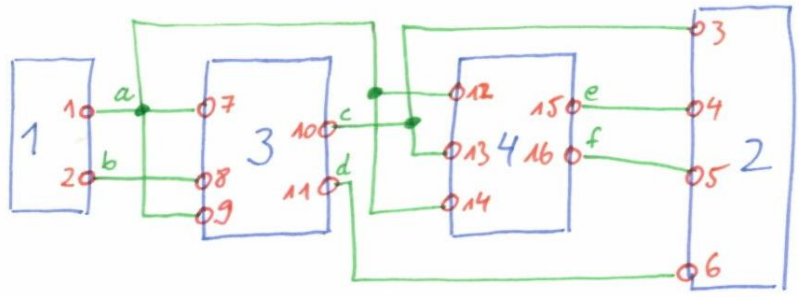


Figure 15: Example of circuit topology for circuit netlist in Figure 14



The reverse relations can also be established:

Eq. 9 **Net-pin relation**: $PN^{-1} = \{(\nu, \rho) | \rho P N \nu\} \subseteq N \times P$

Eq. 10 **Pin-module relation**: $MP^{-1} = \{(\rho, \mu) | \mu M P \rho\} \subseteq P \times M$

Eq. 11 **Net-module relation**: $MN^{-1} = \{(\nu, \mu) | \mu M N \nu\} \subseteq N \times M$

Please note that Figure 14 is the "plotted" graph of the mathematical graph. The corresponding relations of a mathematical graph can be represented on other ways, for instance by an incidence matrix or as a list of incident vertex elements of the graph. For the simplified netlist, we have the following definitions:

Eq. 12 **Net set of module μ**: $N_\mu = \{\nu | (\mu, \nu) \in MN\}$

Eq. 13 **Module set of net ν**: $M_\nu = \{\mu | (\mu, \nu) \in MN\}$

Eq. 14 **Module-net incidence matrix**: $\boldsymbol{MN} = \begin{bmatrix} \vdots & \\ \cdots & MN_{\mu\nu} \quad \cdots \\ \vdots & \end{bmatrix}$, $MN_{\mu\nu} = \begin{cases} 1, & (\mu, \nu) \in MN \\ 0, & \text{else} \end{cases}$

This is illustrated for the example in the following Figure 16:

Figure 16: Module-net list MN, net-module list MN^{-1}, incidence matrix MN for circuit netlist in Figure 14

Module μ	Net set N_μ of module μ
1	a, b
2	c, d, e, f
3	a, b, c, d
4	a, c, e, f

Net ν	Module set M_ν of net ν
a	1, 3, 4
b	1, 3
c	2, 3, 4
d	2, 3
e	2, 4
f	2, 4

$\boldsymbol{MN} =$

	a	b	c	d	e	f
1	1	1	0	0	0	0
2	0	0	1	1	1	1
3	1	1	1	1	0	0
4	1	0	1	0	1	1

Another list of incident vertices would be:

Eq. 15 **Pin set of net ν**: $P_\nu = \{\rho | (\rho, \nu) \in PN\}$

In the example, e.g., $P_a = \{1, 7, 9, 12, 14\}$

2.2 Net models

Net models are established based on exact *pin locations*. Instead of the pin locations, module locations can be used. The *module location* would be for instance taken as the geometrical *center* of the module as determined by the *module dimensions*.

Net models are used as an *approximation of the routing length* that can be computed efficiently and used within the placement process.

Net models usually consist of a *minimum set of two-pin connections* that *spans all adjacent pins or vertices* of the net. There are different models that relate to the technological way of routing. As routing is usually restricted to vertical and horizontal segments, the distance between two pins/module centers is measured in the so-called *Manhattan distance*, which refers to the l_1-norm. The distances can be collected in the *distance matrix*:

Eq. 16 **Distance between pins**:

$$d_{ij} = |x_i - x_j| + |y_i - y_j| \ , \qquad \text{with } (x_i, y_i), (x_j, y_j) \text{ coordinates of pins } i, j$$

Eq. 17 **Distance matrix**:

$$\boldsymbol{D} = \begin{bmatrix} & \vdots & \\ \cdots & d_{ij} & \cdots \\ & \vdots & \end{bmatrix} \quad \text{with } d_{ij} = d_{ji} \quad \text{and } d_{ii} = \infty$$

Each net represents a complete, undirected, weighted distance graph:

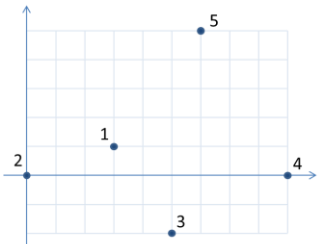
Eq. 18 **Distance graph of net** ν :

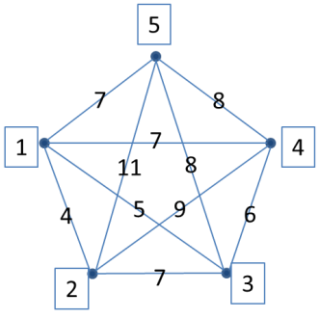
$$G_\nu = \left(P_\nu, P_\nu \times P_\nu, d_{ij}\right)$$

Figure 17 gives an example net with 5 pins.

Figure 17: Example net with pin coordinates and distance matrix

Pin	Pin coordinates
1	(3,1)
2	(0,0)
3	(5,-2)
4	(9,0)
5	(6,5)





$\boldsymbol{D} =$

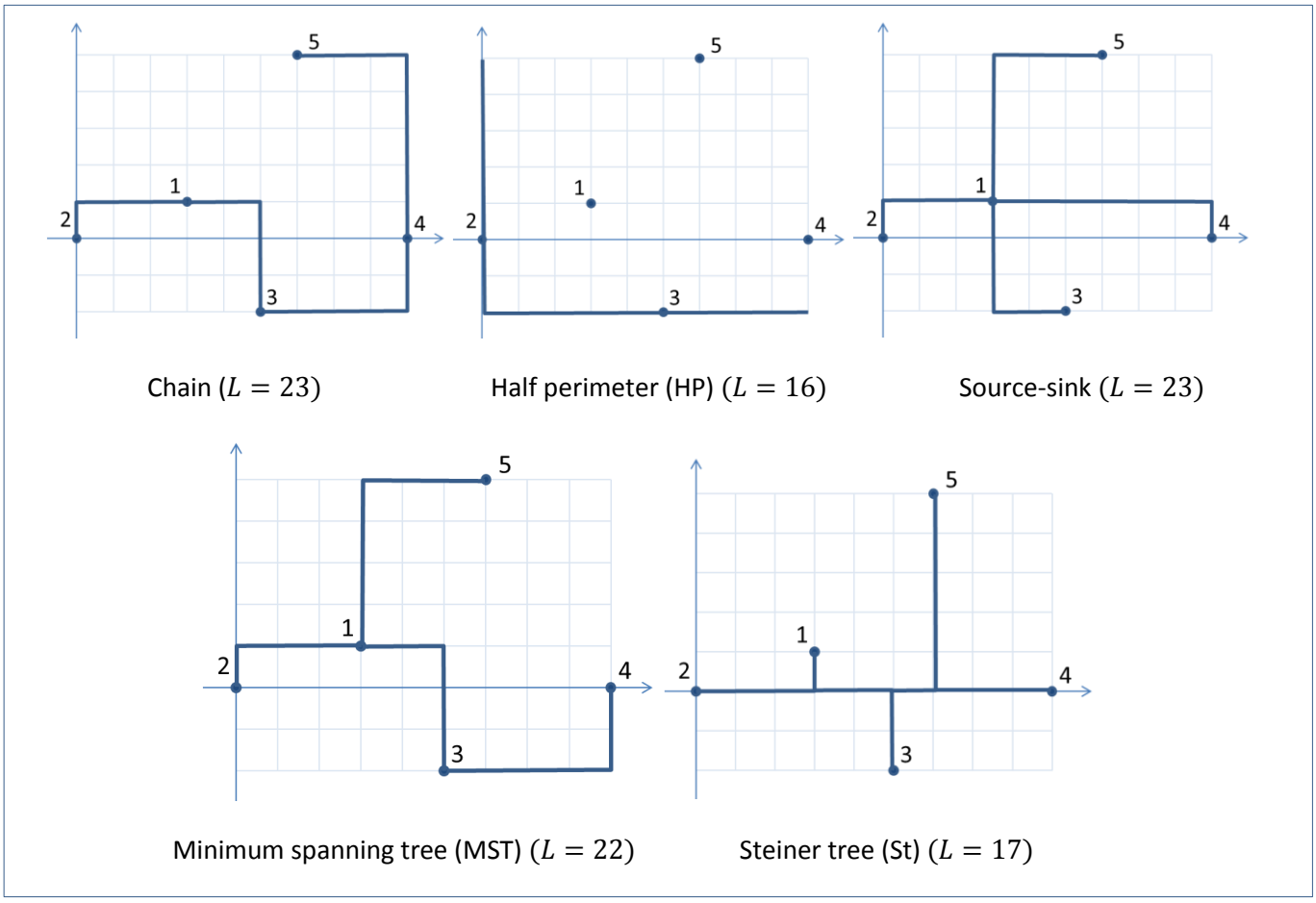
	1	2	3	4	5
1	∞	4	5	7	7
2	4	∞	7	9	11
3	5	7	∞	6	8
4	7	9	6	∞	8
5	7	11	8	8	∞

Typical net models are:

- **Chain net**: This net model refers to the wire-wrap technique. Starting from a certain pin the wire goes from pin to pin without revisiting any pin a second time, thus building a chain to the end pin.
- **Source-sink net**: A certain pin known to be the source of a signal has star-like two-pin connections to every other (sink) pin.
- **Minimum spanning tree (MST)**: An MST has the following properties, it:
  - connects all pins,
  - builds on two-pin connections only,
  - is cycle/loop-free (i.e., tree),
  - has minimum netlength.
- **Minimum Steiner tree (St)**: This is an extension of an MST by allowing virtual pins on already existing two-pin connections that allow a minimum additional routing to remaining pins.
- **Half perimeter (HP) net**: This is a very fast estimation of the routing length through the minimum rectangle enclosing all pins.

Figure **18** shows the five net models for the example of Figure 17 and the corresponding netlengths L:

Figure 18: Net models



2.3 Minimum spanning tree (MST) construction with the KRUSKAL algorithm

A spanning tree of net ν comprises $|P_\nu| - 1$ two-pin connections among the $|P_\nu|$ pins of the net.
There are $(|P_\nu| - 1) + (|P_\nu| - 2) + \cdots = \sum_{i=1}^{|P_\nu|-1} |P_\nu| - i = \frac{1}{2}|P_\nu|(|P_\nu| - 1)$ possible two-pin connections in a complete graph.
The number of possible spanning trees is $|P_\nu|^{|P_\nu|-2}$.

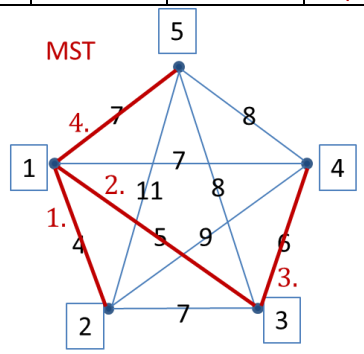
The **basic idea of the Kruskal algorithm** is, to

- order and then select the edges of the MST along increasing edge weights,
- avoid cycles while selecting edges.

For the example in Figure 17 this results in the ordering of edges (along increasing pin numbers) and selection (with skipping one edge, which would lead to a cycle) as given in Figure 19. Please note that the MST in Figure 19 corresponds to Figure 18 bottom row, left.

Figure 19: Kruskal process and resulting MST for example net in Figure 17

edge (i,j)	(1,2)	(1,3)	(3,4)	(1,4)	(1,5)	(2,3)
d_{ij}	4	5	6	7	7	7
Kruskal selection	1.	2.	3.	↯ (cycle)	4.	



Let

Eq. 19 $G_0 = (V_0, E_0, d_{0,ij})$ be an undirected, complete, weighted graph,

Eq. 20 $(Q = \{d_{ij} | (i,j) \in E_0\}, \leq)$ be a **priority queue of ordered edge weights** of G_0,

Eq. 21 $(E, \preccurlyeq)$ be a **priority queue of ordered edges** such that $d(E_i) = Q_i$,

Eq. 22 $G_{MST\mu} = (V_{MST\mu}, E_{MST\mu}, d_{ij})$, $\mu = 1, \dots, n_\mu$, $n_\mu := 0$ be the, initially empty, **set of disjunct partial minimum spanning trees**,

Eq. 23 L_{MST} be the, initially zero, length of the minimum spanning tree $G_{MST} = (V_{MST}, E_{MST}, d_{ij})$.

The Kruskal algorithm is sketched in Figure 20, using Eq. 19 to Eq. 23. The algorithm has to consider that with the addition of new edges to the MST, new partial trees may be opened during the selection process. These partial trees will merge with adequate edges selected at a later time in the process. The existence of partial MSTs needs special consideration to find out if a potential edge would lead to a cycle: this would happen only if both vertices of this edge belong to a single partial MST; if they belong to two different partial MSTs, this would lead to a merger of this partial MSTs.

Figure 20: Kruskal algorithm

Repeat

Select next edge (i^*, j^*) from priority queue, i.e., with minimum weight: $\text{first}\,(Q, \leq) = \min Q = d_{i^* j^*}$

If both vertices i^*, j^* in same partial MST
Goto "Problem reduction"

If none of vertices i^*, j^* in one of the partial MSTs
Open new partial MST, $n_\mu += 1$

If vertices i^*, j^* in two partial MSTs
Merge these partial MSTs, $n_\mu -= 1$

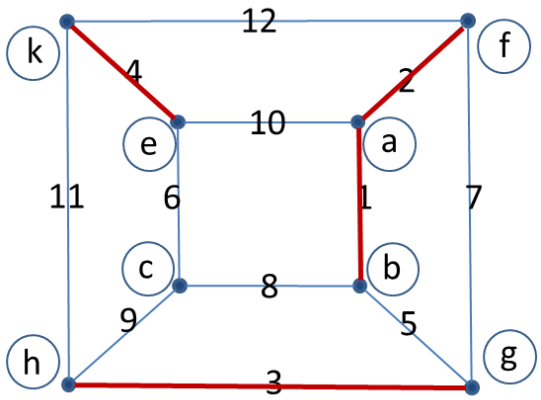
"MST construction":
Attach edge to respective partial MST μ: $V_{MST\mu} \cup = \{i^*, j^*\}$, $E_{MST\mu} \cup = \{(i^*, j^*)\}$, $L_{MST} += d_{i^* j^*}$

"Problem reduction":
$(Q, \leq) \setminus = \{d_{i^* j^*}\}$, $(E, \preccurlyeq) \setminus = \{(i^*, j^*)\}$

Until $\sum_\mu |E_{MST\mu}| = |V_v| - 1$, i.e., the number of edges in the MST is one less than the number of vertices in the graph

The following example illustrates how partial MSTs are created and merged during the Kruskal algorithm. Please note that different to the assumptions for a signal net, this example does NOT represent a complete graph. We could imagine that the non-existing connections refer to forbidden routing areas.

Figure 21: Kruskal algorithm on example with several intermediate partial MSTs



$$(Q, \leq) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

$$(E, \preccurlyeq) = \{(a,b), (a,f), (g,h), (e,k), (b,g), (c,e), (f,g), (b,c), (c,h), (a,e), (h,k), (f,k)\}$$

Step	(i^*, j^*)	L_{MST}	n_μ	$E_{MST\mu}$, $\mu = 1, \ldots, n_\mu$
1	(a,b)	1	1	$\{(a,b)\}$
2	(a,f)	3	1	$\{(a,b),(a,f)\}$
3	(g,h)	6	2	$\{(a,b),(a,f)\}$, $\{(g,h)\}$
4	(e,k)	10	3	$\{(a,b),(a,f)\}$, $\{(g,h)\}$, $\{(e,k)\}$ (MSTs given above in red)
5	(b,g)	15	2	$\{(a,b),(a,f),(b,g),(g,h)\}$, $\{(e,k)\}$
6	(c,e)	21	2	$\{(a,b),(a,f),(b,g),(g,h)\}$, $\{(c,e),(e,k)\}$
7	~~(f,g)~~	21	2	↯ (cycle)
8	(b,c)	29	1	$\{(a,b),(a,f),(b,c),(b,g),(c,e),(e,k),(g,h)\}$

While the Kruskal algorithm proceeds along edges, we can also proceed along vertices. This is done by the Prim algorithm presented in the following.

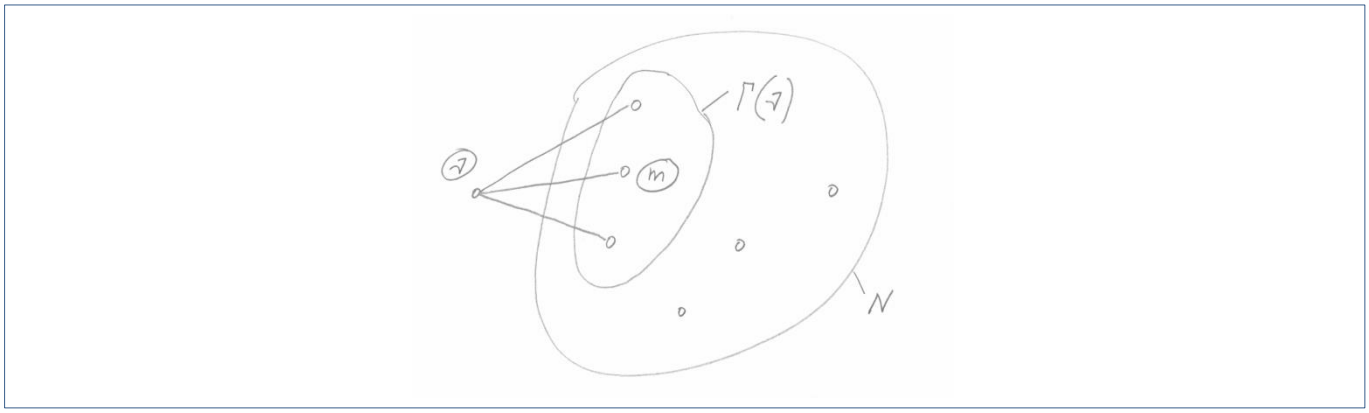
2.4 Minimum spanning tree (MST) construction with the PRIM algorithm

The starting point of the PRIM method to construct a minimum spanning tree is the following theorem:

Theorem 1 The edge from a vertex v to its neighbor m (see Figure 22) with minimum corresponding edge weight d_{vm} is part of the minimum spanning tree:

$$(v,m) = \text{argmin}_{(v,j)}\{d_{vj} | j \in \Gamma(v)\} \Rightarrow (v,m) \in E_{MST}$$

Figure 22: Neighbors Γ of vertex v in set N



Theorem 1 allows the **construction of a minimum spanning tree** by

- starting from any initial vertex,
- searching among his neighbors the edge with minimum weight,
- "merge" the vertices,
- "merge" the edges and updating the edge weights (take minimum among alternative).

The PRIM process will be illustrated with an example in the following (Figure 23), after that the algorithm will be given. Explanation of Figure 23:

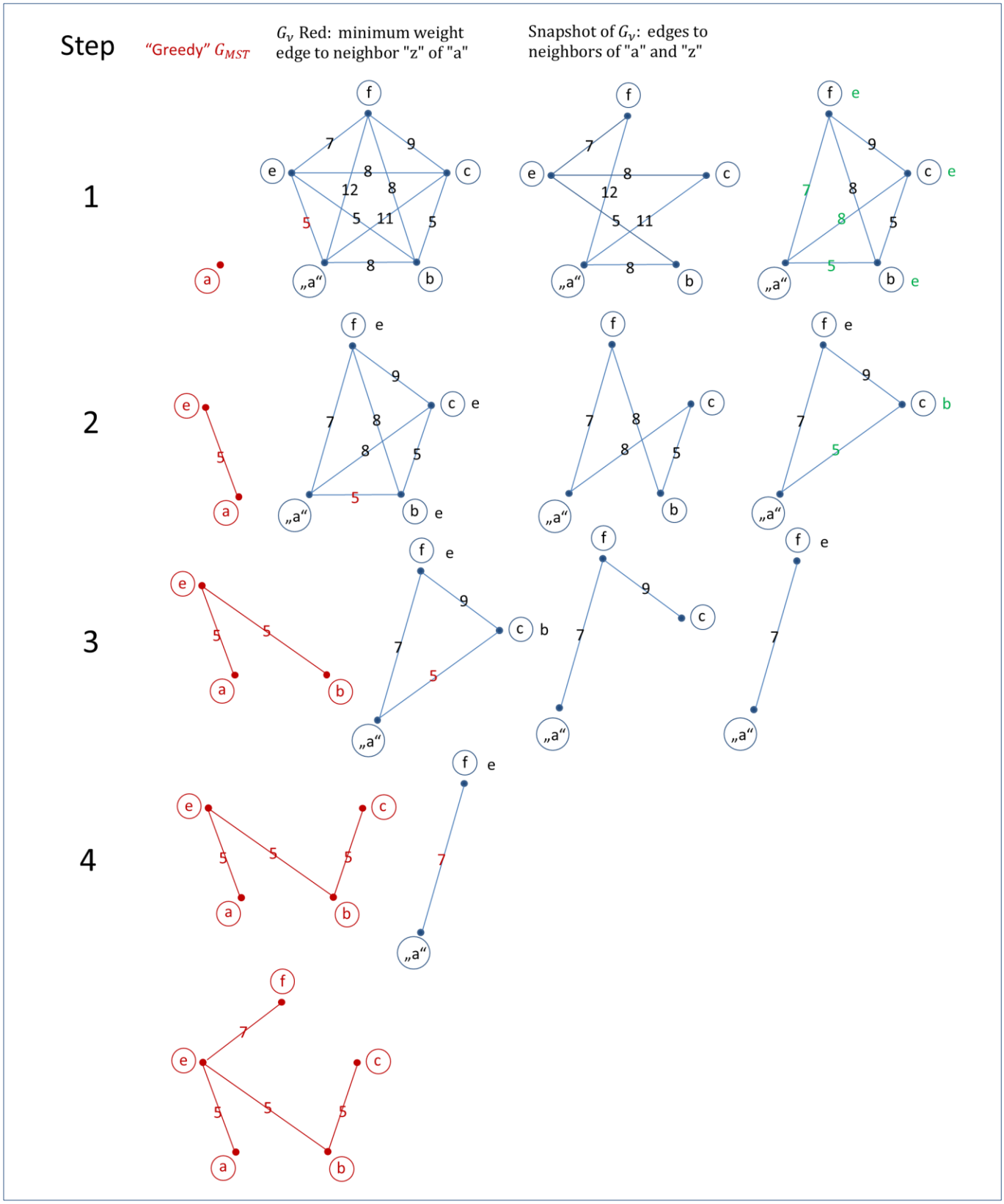
Step 1: We start with an arbitrary initial minimum spanning tree consisting of one vertex. Here, we take vertex a. Then, all neighbors of vertex a are considered and the vertex with minimum edge weight is selected. According to Theorem 1, this edge must be part of the MST. Here we select vertex e, as the edge (a,e) has minimum weight among all edges to a's neighbors:

$$(a,e) = \text{argmin}_{(a,j)}\left\{d_{aj} \middle| j \in \Gamma(a) = \{b,c,e,f\}\right\} = \text{argmin}\{8,11,5,12\}$$

The corresponding edge weight is marked in red in Figure 23, third column. The selected vertex gets an intermediate variable name, "z", and it is added to the MST (step 2, second column). At this stage, the MST becomes:

$$V_{MST} = \{a,e\},\ E_{MST} = \{(a,e)\},\ L_{MST} = 5$$

Figure 23: Example process of MST construction with the Prim method



The remaining graph G_v now is reduced of the selected vertex z. This can be interpreted as merging vertex z with the existing vertex a and creating a new super node "a". The PRIM algorithm is a so-called "greedy" algorithm, as it subsequently "swallows" all vertices of the graph.

While swallowing vertex z, two actions have to be taken:

1. Among the remaining vertices, there are multiple edges to the new super node a. Among those the one with the minimum edge weight must be selected, and the other ones will be deleted. This reduces the problem size to those edges which can only be candidates for the MST. Column 4 in Figure 23 gives a snapshot of the graph with edges from the super node to the remaining nodes. In this first step of the PRIM process, we can see that edge (e,f) has a smaller edge weight than (a,f), (e,c) than (a,c), (e,b) than (a,b). After reduction of vertex z and of the edges with larger weights, we obtain the new remaining graph in column 5. The new edge weights are marked in green.
2. To save computational time in the further process, it should be stored which respective neighbor of each vertex in the remaining graph actually leads to the new edge weight. This is done by giving the nodes labels that refer to the neighbor in the super node with that edge weight. In this step, this is node e for all remaining nodes b, c, f. These labels are marked in green in column 5 of step 1.

The process is now repeated.

Step 2: Vertex b is selected, as edge (a,b) has minimum weight among all edges to a's neighbors:

$$(a,b) = \text{argmin}_{(a,j)}\left\{d_{aj}\middle| j \in \Gamma(a) = \{b,c,f\}\right\} = \text{argmin}\{5,8,7\}$$

As the label of vertex b says that this edge is to vertex e , the MST becomes:

$$V_{MST} = \{a,b,e\},\ \ E_{MST} = \{(a,e),(b,e)\},\qquad L_{MST} = 10$$

Checking the multiple edges from the new super node a that swallows b (step 2, column 4 of Figure 23), we see that the edges (a,f) and (b,c) have smallest weight. The other edges are deleted, and a new edge weight and label to vertex c are obtained. This results in the remaining graph in step 2, column 5, of Figure 23.

Step 3: Vertex c is selected, as edge (a,c) has minimum weight among all edges to a's neighbors:

$$(a,c) = \text{argmin}_{(a,j)}\left\{d_{aj}\middle| j \in \Gamma(a) = \{c,f\}\right\} = \text{argmin}\{5,7\}$$

As the label of vertex c says that this edge is to vertex b , the MST becomes:

$$V_{MST} = \{a,b,c,e\},\ \ E_{MST} = \{(a,e),(b,e),(b,c)\},\qquad L_{MST} = 15$$

Checking the multiple edges from the new super node a that swallows c (step 3, column 4 of Figure 23), we see that the edges (a,f) has smallest weight. The other edge is deleted, this results in the remaining graph in step 3, column 5, of Figure 23.

Step 4: Only one vertex remains, which is selected, which leads to the final MST:

$$V_{MST} = \{a,b,c,e,f\},\ \ E_{MST} = \{(a,e),(b,e),(b,c),(e,f)\},\qquad L_{MST} = 22$$

The algorithm that implements this process is sketched in Figure 24.

Figure 24: Prim algorithm

Initially, $V_{MST} := \{a\}$ (arbitray), $E_{MST} := \{\ \}$, $L_{MST} := 0$, $\forall_{n \in \Gamma(a)} \text{label}(n) := a$, $V := V_0$, $E := E_0$, $d_{ij} = d_{0,ij}$

Repeat

Select next vertex z with minimum edge weight d_{az} (sorting algorithm):

$$(a, z) = \text{argmin}_{(a,n)}\{d_{an} | n \in \Gamma(a)\}$$

"MST construction":

Add new vertex and edge to MST:

$$V_{MST} \cup = \{z\}, \quad E_{MST} \cup = \{(z, \text{label}(z))\}, \quad L_{MST} += d_{az}$$

(label(z) denotes the neighbor of z that corresponds to the minimum edge weight d_{az})

"Edge revision for merger of a, z":

For all neighbors n of vertex z except a : $n \in \Gamma(a) \setminus \{a\}$

If edge (a, n) did not exist

Create edge (a, n): $E := E \cup \{(a, n)\}$; $d_{an} := d_{zn}$; $\text{label}(n) := z$

Else (check if new edge from a to n induced by z has lower weight than old edge)

If $d_{zn} < d_{an}$

Update edge weight and label: $d_{an} := d_{zn}$; $\text{label}(n) := z$

Endif

Endif

Endfor

"Problem reduction":

Delete "swallowed" vertex z and all its adjacent edges from graph:

$E := E \setminus \{ (z, n) \mid n \in \Gamma(z) \}$, $V = V \setminus \{z\}$

Until $|E_{MST}| = |V_0| - 1$, i.e., the number of edges in the MST is one less than the number of vertices in the graph

2.5 Steiner tree (St) approximation with the HANAN algorithm

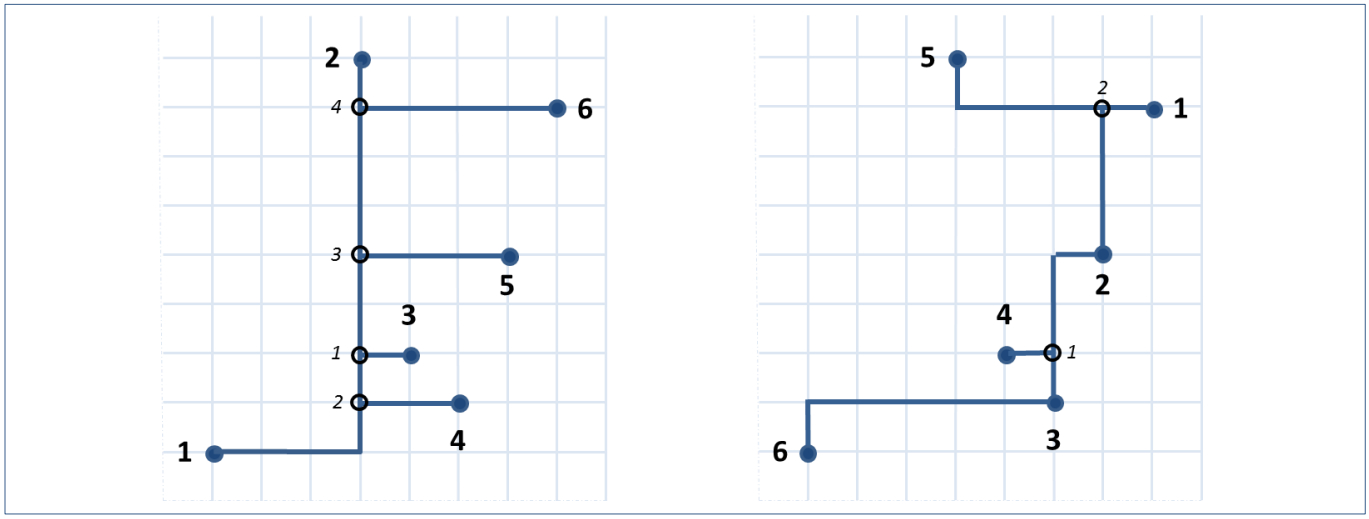
A Steiner tree is a net that connects all pins without any cycles. Specifically, it allows that additional "Steiner pins" are added such that the overall routing lengths is smaller than with a minimum spanning trees (which allows only two-pin connections between existing pins).

The **Hanan algorithm** (M. Hanan: On Steiner's Problem with Rectilinear Distances, J. SIAM: Appl. Math., Vol. 14. No. 2, 1966: M. Servit: Heuristic Algorithms for Rectilinear Steiner Trees, Digital Processes, 7 (1981), pp. 21-32) for rectilinear Steiner trees is a **heuristic with the following features**:

1. The pins are ordered and processed along **increasing x-coordinates** (i.e., from left to right).
2. Each pin is connected to the net tree constructed so far by finding the **shortest Manhattan route/distance with at most one corner to any point of the net tree**. If this point is not a pin, a new pin ("Steiner pin") is created at this point.
3. New routes are preferably **horizontal** (—) or **vertical** (|) without corner, if that is not possible, new routes have a **corner at the x-coordinate of the new pin that is connected up or down** (↥ or ↧).

The left side of Figure 25 illustrates an example for a Steiner tree with the Hanan method. In the first step, pin 2 is connected to pin 1. Then, pin 3 is connected by a 1-grid long horizontal route the existing net, and the new Steiner pin *1* is created. Here, the priority list in the route shapes is illustrated: from left to right, a route should have a horizontal segment before a corner and then a vertical segment, because the vertical might become a potential docking station for pins further to the right. If this vertical part would be far left, the next routes would be longer.

Figure 25: Steiner tree approximation with Hanan algorithm, from left to right (on the left side), from right to left (on the right side), Pins (filled) are numbered in the order of processing, Steiner points (not filled) are numbered in the order of their creation



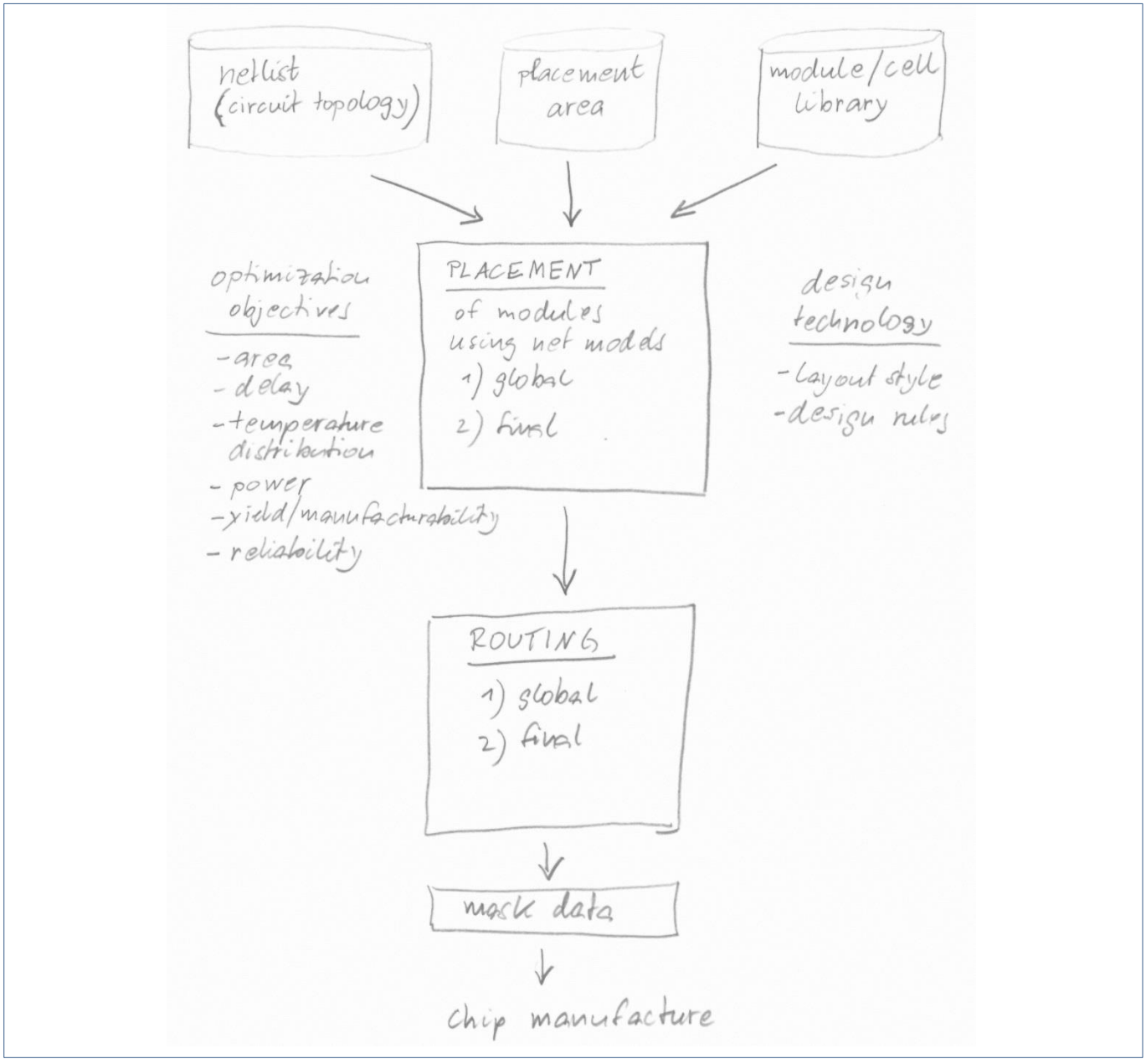
Please note that a minimum rectilinear Steiner tree can be constructed by creating a grid through each of the pins (solid lines in Figure 25). It can be shown that the routing segments of the Steiner tree will be on this grid only (there are no routing segments of the Steiner tree on dotted lines in Figure 25).

The algorithm is a heuristic that cannot guarantee to find a minimum Steiner tree. See for instance the right side of Figure 25, where the Hanan algorithm is applied to the same net, this time from right to left, with analogous rules. The Steiner tree on the left side has a length of 21, the one on the right side of 19.

3. Placement problem description

Figure 26 shows the principal layout design flow. It consists of the steps placement and routing. During placement, net models are used to approximate the routing effects of the layout for the placement decisions (complete routing would be too expensive in complex systems). For complexity reasons, placement and routing each are divided into a global and final step.

Figure 26: Layout design flow



In global placement, the relative placement of modules is determined, where the modules may still overlap. A common approach for global placement is quadratic placement (also called force-directed placement).

Quadratic placement formulates an iterative process with a quadratic optimization problem with equality constraints in each iteration step. In each iteration step, this leads to a (large, sparse) linear equation system that has to be solved.

The placement task can be formulated in two flavors:

3.1 Assignment

Assignment starts from a given set of positions, where the number of available positions has to be greater or equal than the number n_M of modules. The assignment task consists in assigning modules to the available ("discrete") positions, such that the circuit can be routed and with minimum net length. This is illustrated in Figure 27.

This task refers to the so-called gate array design style, which is a special case of the standard cell design style in Section 1.1.

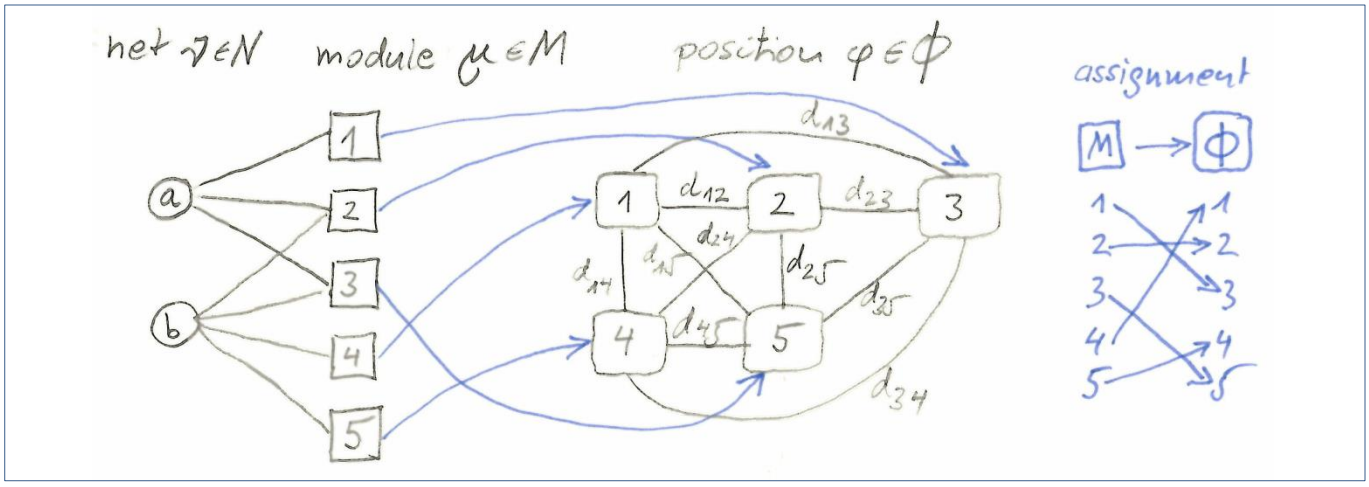
The complexity of this task is $n_M!$, as there are $n_M!$ alternatives to assign n_M modules to n_M positions. This problem is called NP hard, i.e. it is not solvable in polynomial time.

Assignment can be formulated as a permutation, as illustrated in the right part of Figure 27, where a set of module indices maps onto a set of position indices: $(1,2,3,4,5) \subset M \rightarrow (3,2,5,1,4) \subset \Phi$. We can formulate the module-position relation of assignment as:

Eq. 24 **Module-position relation** $M\Phi = \{(\mu,\varphi)|\mu M \Phi \varphi\} \subseteq M \times \Phi$

In the example, we have $M\Phi = \{(1,3),(2,2),(3,5),(4,1),(5,4)\}$.

Figure 27: Assignment task, exemplified with 5 modules and 5 locations



Different sizes of modules need special treatment in assignment (e.g., partitioning into smaller parts that fit into the grid). The minimum area objective is inherent in the minimum net length objective; requirements on aspect ratio and dimensions of the resulting array need consideration (e.g., by corresponding sets of available positions).

3.2 Arrangement

Arrangement has more degrees of freedom in the placement task: the modules have different sizes, the positions and the orientations of the modules may be freely chosen. The arrangement task consists in finding "continuous" positions, such that the circuit can be routed and with minimum area and minimum net length

This task refers to the macro cell design style in Section 1.1. If the arrangement is done for a high-level system description where the modules refer to "big" macros like a whole standard cell block, a RAM block, and so one (see Figure 13), then this task is referred to as "**floorplanning**". If the arrangement is done for a detailed system description, where the modules refer for instance to all cells in the standard cell block, then this task belongs to the category of placement.

In the following, assignment will be treated first, then arrangement. For both approaches, a suitable net model has to be developed. This is done in the following section.

3.3 Hyperedge net model/nth-order assignment problem

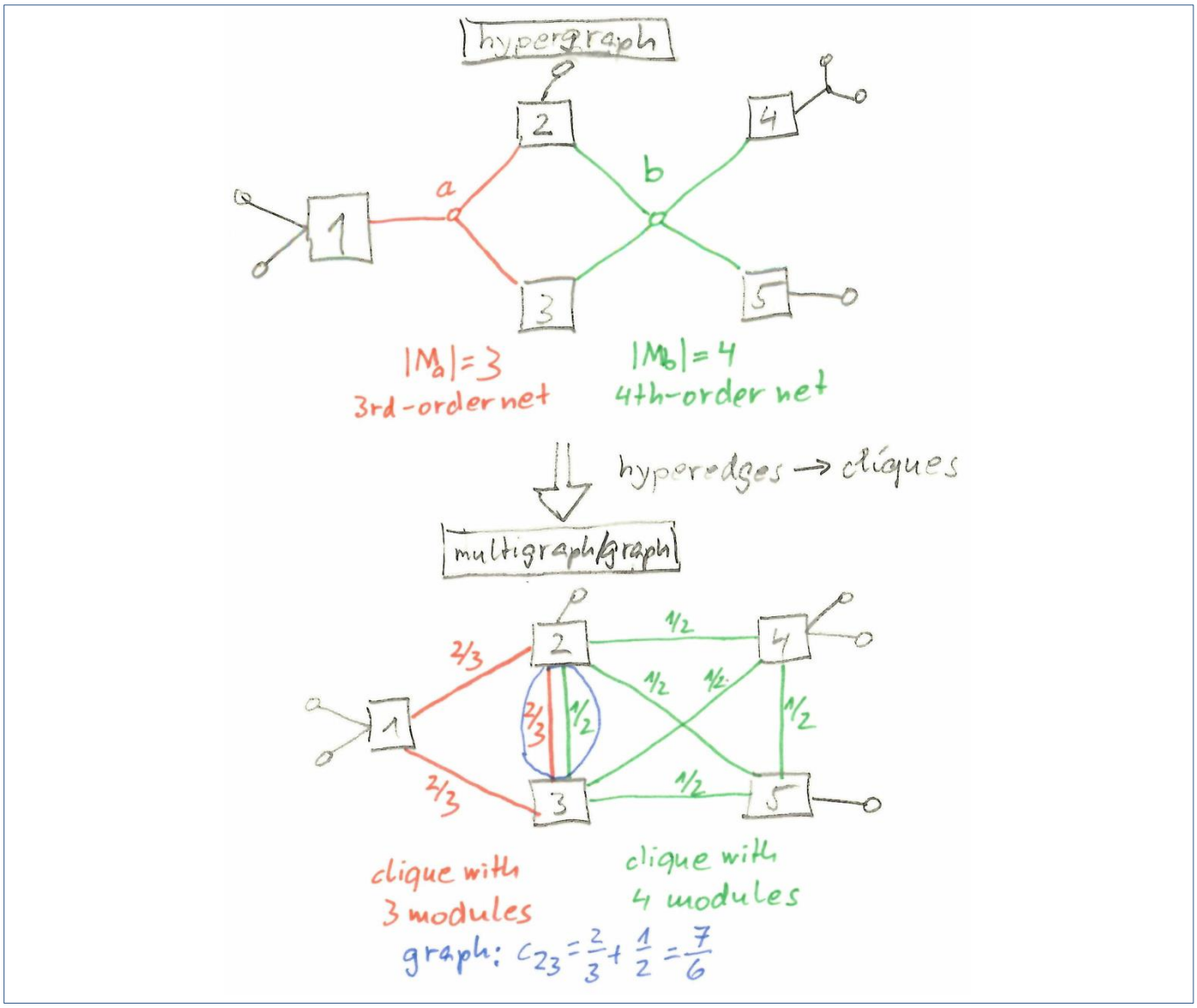
The general description of the circuit topology from the netlist according to Section 2.1 leads to a **hypergraph** as illustrated in the top part of Figure 28. A hypergraph is characterized by **hyperedges**, these are edges that connect more than two vertices. This corresponds to the property that signal nets connect several modules. The number of vertices/modules that are connected with a hyperedge/net is called the **order of a hyperedge**/net. The maximum order hyperedge determines the order of the resulting assignment problem. In the example of Figure 28, there is one 3rd-order net and one 4th-order net, hence a 4th-order assignment problem.

The nth-order assignment problem then has the objective of minimizing the sum of all net lengths $L(v), v \in N$ of a circuit. (The same problem formulation refers to the arrangement problem, if the net length is taken as primary objective.)

Eq. 25 **n^{th}-order assignment problem:** (with $n = \max_v |M_v|$)

$$\text{minimize}_{M\Phi} \sum_{v \in N} L(v)$$

Figure 28: Circuit topology – from hypergraph of 4th-order to weighted graph



3.4 Clique net model/2nd-order(quadratic) assignment problem

The hypergraph model of circuit topology is difficult to handle, therefore a transformation to a regular graph is done. Each hyperedge is modeled as a clique of edges among all vertices in the respective hyperedge. This leads to a multigraph in the first place, as illustrated in the lower part of Figure 28. The edges are weighted in order to reflect the various available orders of nets and the consequences on routing. A common idea for this is starting from the minimum spanning tree. A net of order m has a minimum spanning tree with $m-1$ two-pin connections. A simple model for the netlength of a net of order n is this number of two-pin connections. Hence the weight of a hyperedge of order m is assumed to be $m-1$. This is transformed into the graph model in that the sum of all $\frac{m\cdot(m-1)}{2}$ edge weights of a clique should also have the value $m-1$. This is achieved by:

Eq. 26 **edge weight** $\frac{2}{m}$ for each edge of a clique that refers to a **net of order** m.

The transition from a multigraph to a graph is done by adding the weights if several edges between two modules exist and replace these edges with one edge with the resulting sum of weights (blue in Figure 28).

The 2nd-order assignment problem then consists in minimizing the weighted distances d between the positions $\varphi(i), \varphi(j)$ of all modules i, j, where the weights are determined according to the signal nets as described:

Eq. 27 **2nd-order/quadratic assignment problem:**

$$\text{minimize}_{M\Phi} \sum_{i,j\in M} \frac{1}{2} C_{ij} \cdot d^2_{\varphi(i),\varphi(j)}$$

The term "2nd order" refers to the existence of edges of 2nd order between two modules only. The term quadratic refers to the objective function in Eq. 27, which can be formulated as a quadratic form (see XXX).

The quadratic *placement* problem will be formulated in XXX. Quadratic assignment/placement is complex, as both relations among modules (edge weights) and relations among positions (distance) go into the objective function. They are interdependent.

Another approach to assignment is to partition the problem such that the assignment of a module to a position is independent of all other assignments. This is called linear assignment, as it can be formulated as a linear programming problem. Linear assignment will be treated in the following.

3.5 Linear assignment problem formulation

The linear assignment problem consists in minimizing the sum of costs of assigning modules to positions. These costs shall be independent of each other. Then, the optimization problem becomes a linear programming problem:

Eq. 28 **Linear assignment problem:**

$$\text{minimize}_{M'\Phi} \sum_{i \in M'} k_{i,\varphi(i)}$$

where $\boldsymbol{K} = [k_{i,j}]$ is the cost matrix with $k_{i,j}$ being the **cost (=routing length)** of assigning module i to position j,

and where $M' \subseteq M$ is a subset of modules for which the cost of assigning one module to a position is independent of the assignments of the other modules in M'.

The approach requires the computation of subsets of independent modules in the sense that the assignment of one module in the subset does not affect the routing cost of any other module in the subset. This is the case if the modules in the subset are **not connected** by a net. Then, the netlength of assigning one independent module does not depend on the assignment of the other independent modules in the same subset.

Introducing an example to illustrate linear assignment

Assignment knows different types of modules. These are defined in the following

Eq. 29 **Modules to be assigned**: $M_{\square}$

Eq. 30 **Modules with predefined positions**: $M_{\odot}$

Eq. 31 **Auxiliary modules** : $M_{\bullet}$

An introductory example that will be used to demonstrate the linear assignment algorithm is given in the upper part of Figure 29. It consists of 5 modules to be assigned to 8 available positions, while 2 modules have predefined positions. One auxiliary module is added, which has no connection to any other module and acts as a dummy to have equal number of modules and positions. The net-module and module-net lists are given

The assignment cost is determined by constructing a minimum spanning tree (MST) for each net. The length of the MST of each net is obtained by counting the jumps in Manhattan geometry from one position to another.

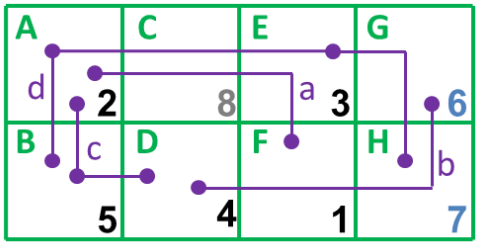
The lower left part of Figure 29 shows an initial assignment of the modules to a given 2x4 grid of positions, as well as the resulting MSTs plus their lengths and the overall length (=cost of initial assignment).

Figure 29: Net-module list MN^{-1}, module-net list MN, initial assignment and routing cost of an assignment example

Net $\nu \in N$	Module set M_ν of net ν
a	1, 2
b	4, 6
c	2, 4, 5
d	2, 3, 5, 7

Module $\mu \in M$	Net set N_μ of module μ
1	a
2	a, c, d
3	d
4	b, c
5	c,d
6	b
7	d
8	-

Assignable: $M_\square = \{1, 2, 3, 4, 5\}$
Predefined: $M_\odot = \{6, 7\}$
Auxiliary: $M_\bullet = \{8\}$



Net $\nu \in N$	Minimum spanning tree (MST) length $L(\nu)$
a	3
b	3
c	2
d	5
$\sum_{\nu \in N} L(\nu)$	13

4. Linear assignment of independent module subsets (STEINBERG method)

Figure 30 gives an overview of the Steinberg method for linear assignment. The individual steps will be described in more detail in the subsequent sections.

Figure 30: Steinberg method for linear assignment

Determine all maximum subsets $C_i, i = 1, \dots, n_C$ of independent modules (i.e., cliques in $G_{\overline{V}}$)
// **Section 4.1**
//
Determine initial assignment
//
Repeat
    For each subset C_i
        Set up cost matrix $\boldsymbol{K}$ for all modules μ in C_i and all positions $\varphi(\mu)$ these modules currently have

$$k_{\mu,\varphi(\mu)} = \sum_{\nu \in N} L(\nu)$$

        // **Section 4.2**
        // Note that only nets that are attached to a considered module need to be considered to
        // compute a change in cost function, as the lengths of other nets are not affected by a
        // repositioning this module
        //
        // Note also that each element $k_{\mu,\varphi(\mu)}$ in the cost matrix can be determined independently from
        // the other modules in subset C_i, as the corresponding module is only connected to modules
        // outside of this subset
        //
        Determine new assignment with minimum cost

$$\sum_{\mu \in C_i} k_{\mu,\varphi(\mu)}$$

        by Hungarian method
        // **Section 4.3**
        // This is the mathematical core of linear assignment.
    Endfor
Until no further improvement in cost

4.1 Determine maximum independent module subsets (maximum cliques in a graph)

Eq. 32 **Graph of assignable modules connected by a net**:

$$G_{cy} = (M_{\square}, MN \circ MN^{-1} \cap (M_{\square} \times M_{\square} \setminus I))$$

Eq. 32 describes mathematically the graph of connected assignable modules by taking the relation from modules to nets followed by the relation from nets to modules, $MN \circ MN^{-1}$, and taking only the combinations of non-identical assignable modules, $M_{\square} \times M_{\square} \setminus I$.

Eq. 33 **Graph of assignable modules NOT connected by a net**:

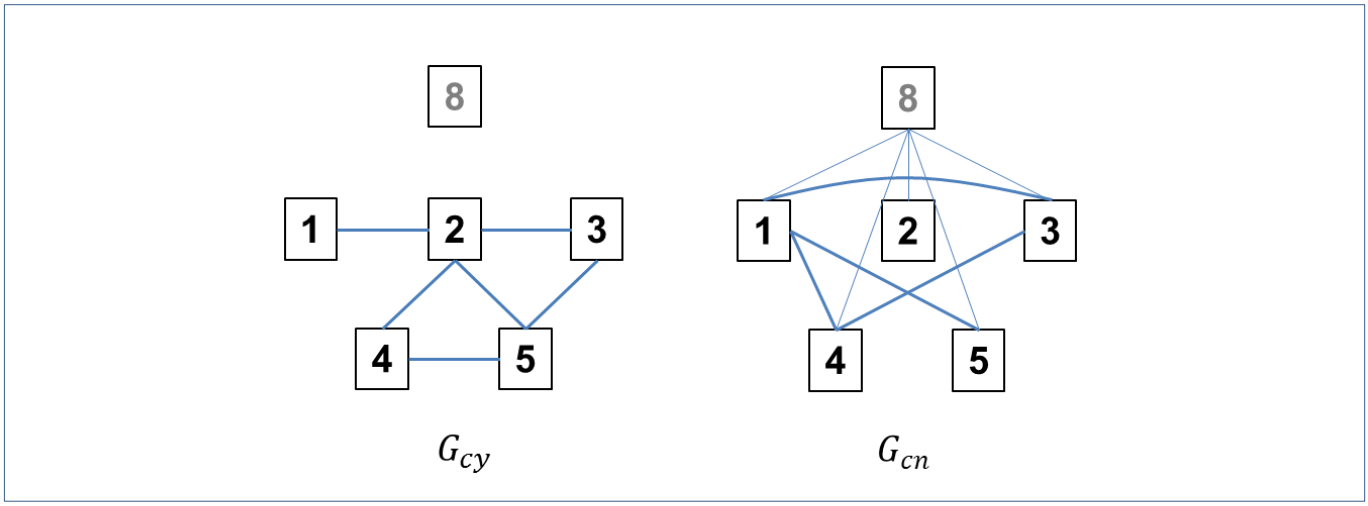
$$G_{cn} = \left(M_{\square}, \overline{MN \circ MN^{-1}} \cap (M_{\square} \times M_{\square} \setminus I)\right)$$

Eq. 33 describes mathematically the graph of unconnected assignable modules by taking the inverse of the relation from modules to nets followed by the relation from nets to modules, $\overline{MN \circ MN^{-1}}$, and taking only the combinations of non-identical assignable modules, $M_{\square} \times M_{\square} \setminus I$.

For the example in Figure 29 we obtain the graphs G_{cy} and G_{cn} as shown in Figure 31. We can note several observation for these graphs:

- The modules with predefined positions do not appear.
- The auxiliary module has no adjacent edge in G_{cy}, and it has an edge to every vertex in G_{cn} .
- The graph G_{cy} can be constructed from the net-module list (e.g. top left in Figure 29): each net represents a hyperedge among the modules it connects, which transforms into a so-called clique, which is a subgraph where every vertex has an edge to every other vertex.

Figure 31: Graph G_{cy} of connected and graph G_{cn} of unconnected modules of the example in Figure 29



Eq. 34 A **Clique** $C = (V_C, E_C)$ of a graph $G = (V_G, E_G)$ is a:

complete, i.e., edges between every pair of vertices: $E_C = (V_C \times V_C) \setminus I$,
subgraph, i.e., $V_C \subseteq V_G$ of G

A clique is described by its vertex set only, hence it can be noted in a simplified way: $C \equiv V_C$

While the cliques in G_{cy} are obvious from the net-module list, for assignment we are rather interested in the (maximum) cliques of G_{cn} . In Figure 31 for instance, we can recognize the following maximum cliques:

- $\{1, 3, 4, 8\}$
- $\{1, 5, 8\}$
- $\{2, 8\}$

This means that we can consider a swap for instance among the modules 1, 3, 4, and 8 (8 basically represents a Joker position that can be occupied in any case), where the cost of assigning for instance module 1 to one of the positions that 3, 4, 8 currently take, can be determined without knowing in advance where these other modules will be positioned. Because 1 has no connection with 3, 4, 8, it is only connected to modules “outside” the clique, which are fixed at this point of consideration, so that the netlength is determined the moment when 1 is positioned among the available positions of the considered clique.

Theorem 2 (**Paull/Unger) for iterative construction of cliques of a graph**:

$$\mathbb{C}_V = \left\{ C = (V_C, E_C) \;\middle|\; \underbrace{V_C \in P(V) \setminus \{\}}_{\substack{\text{power set of } V, \text{ i.e., all subsets} \\ \text{including nondominant cliques}}} \;\wedge\; \underbrace{E_C = (V_C \times V_C) \setminus I}_{\text{complete graph}} \right\}$$ is the set of all cliques

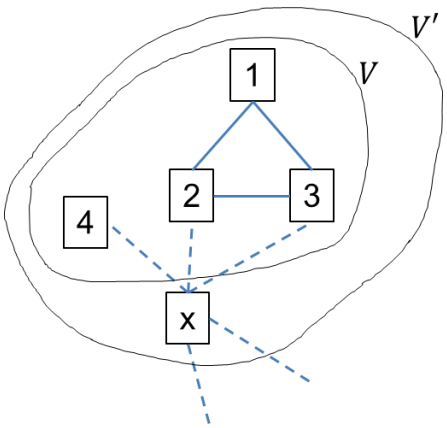
of graph $G = (V_G, E_G)$ with regard to vertex subset $V \subset V_G$.

Then, the set of all cliques of graph G with regard to a vertex subset extended by one vertex x, i.e., $V' = V \cup \{x\}, x \in V_G \setminus V$, is:

$$\mathbb{C}_{V'} = \mathbb{C}_V \cup \bigcup_{V_C \subset \mathbb{C}_V} \left\{ V_{C'} \;\middle|\; V_{C'} \in P(V') \setminus \{\} \;\wedge\; V_{C'} = \underbrace{\{x\}}_{\substack{\text{vertices from remaining} \\ \text{set of nodes in } N_G}} \cup \left[\underbrace{\Gamma(x)}_{\text{neighbors of } x} \cap \underbrace{V_C}_{\substack{\text{that are in} \\ \text{clique of } \mathbb{C}_N}} \right] \right\}$$

Theorem 2 says that the cliques of a graph can be constructed starting from any initial single clique or subset of cliques (which includes to start from a single node), and then iteratively adding a remaining node x and the resulting addition of new cliques that result from combining x with its neighbors in existing cliques.

Figure 32: Illustration of Theorem 2 (Paull/Unger)



$$\mathbb{C}_V = \{\{1,2,3\},\{4\}\}$$

$$\mathbb{C}_{V'} = \mathbb{C}_V \cup \{\{x\} \cup [\{2,3,4,\dots\} \cap \{1,2,3\}]\} \cup \{\{x\} \cup [\{2,3,4,\dots\} \cap \{4\}]\}$$

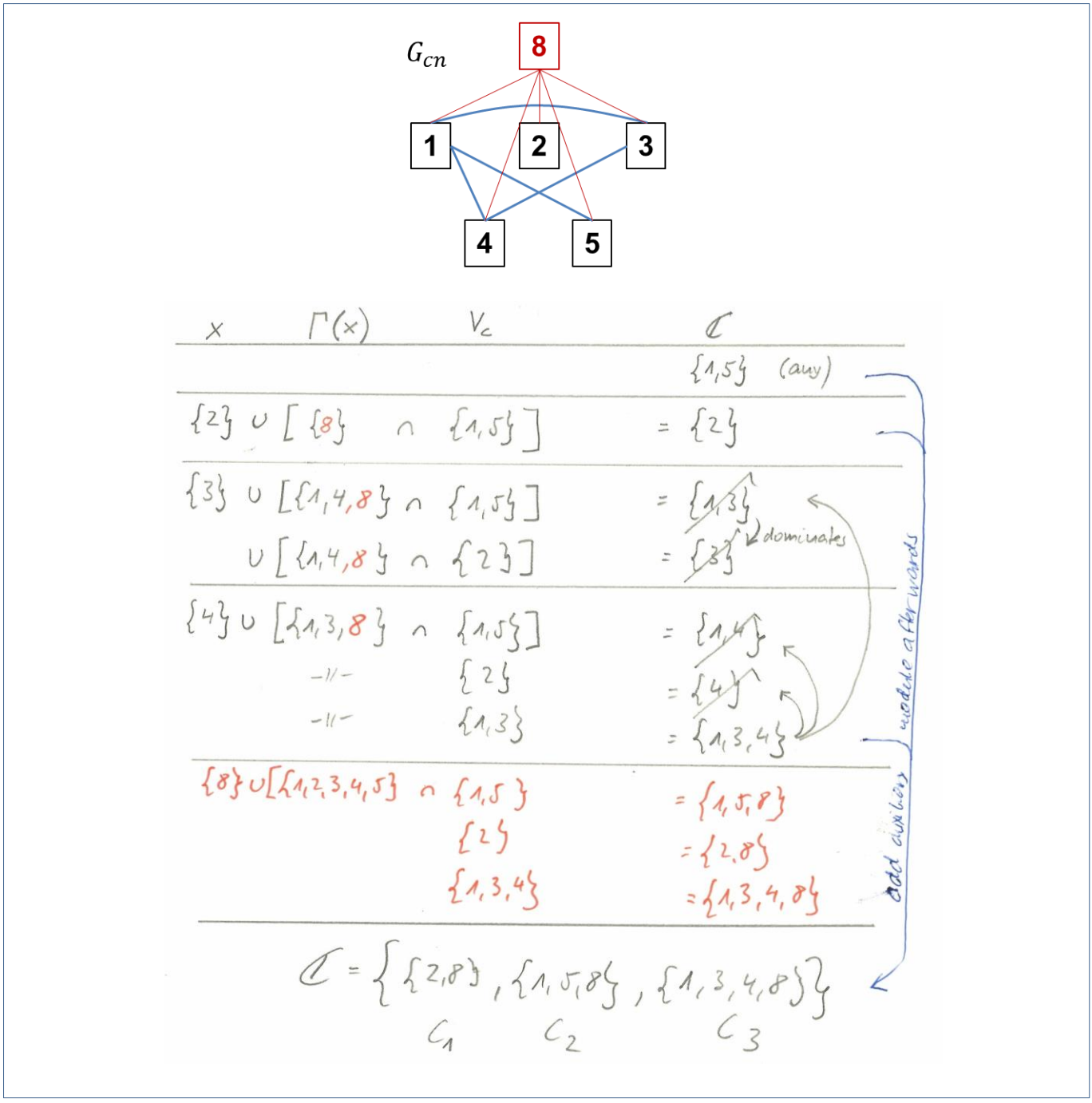
$$= \mathbb{C}_V \cup \{x,2,3\} \cup \{x,4\}$$

$$= \{\{1,2,3\},\{x,2,3\},\{x,4\}\}$$

Figure 32 illustrates the procedure of **Theorem** 2. At the moment, the vertex set {1, 2, 3, 4} refers to cliques {1, 2, 3} and {4}. Adding vertex {x}, we determine the cut sets of the neighbors of x and these cliques and the node x to obtain the additional cliques {x, 2, 3} and {x, 4}. Please note that in the overall process, it is only necessary to keep the dominant cliques. Cliques covered by dominant cliques need not be mentioned separately. For instance, the dominant clique {x, 4} covers clique {4}.

Figure 33 then illustrates the complete process of constructing the cliques of unconnected modules of the assignment problem in Figure 29. The auxiliary module 8 is ignored. It can be seen that including 8 into the process does not yield any change in the intermediate results until module 8 is added. We can see that an auxiliary module can be ignored throughout the process, and just added to all cliques at the end. This corresponds to the fact that an auxiliary module basically represents an available position and is never connected to any real module. At this point of the assignment method according to Figure 30, we have three cliques of unconnected modules, i.e., {2, 8}, {1, 5, 8} and {1, 3, 4, 8}, each of which represents a set of modules whose assignment costs are not interdependent.

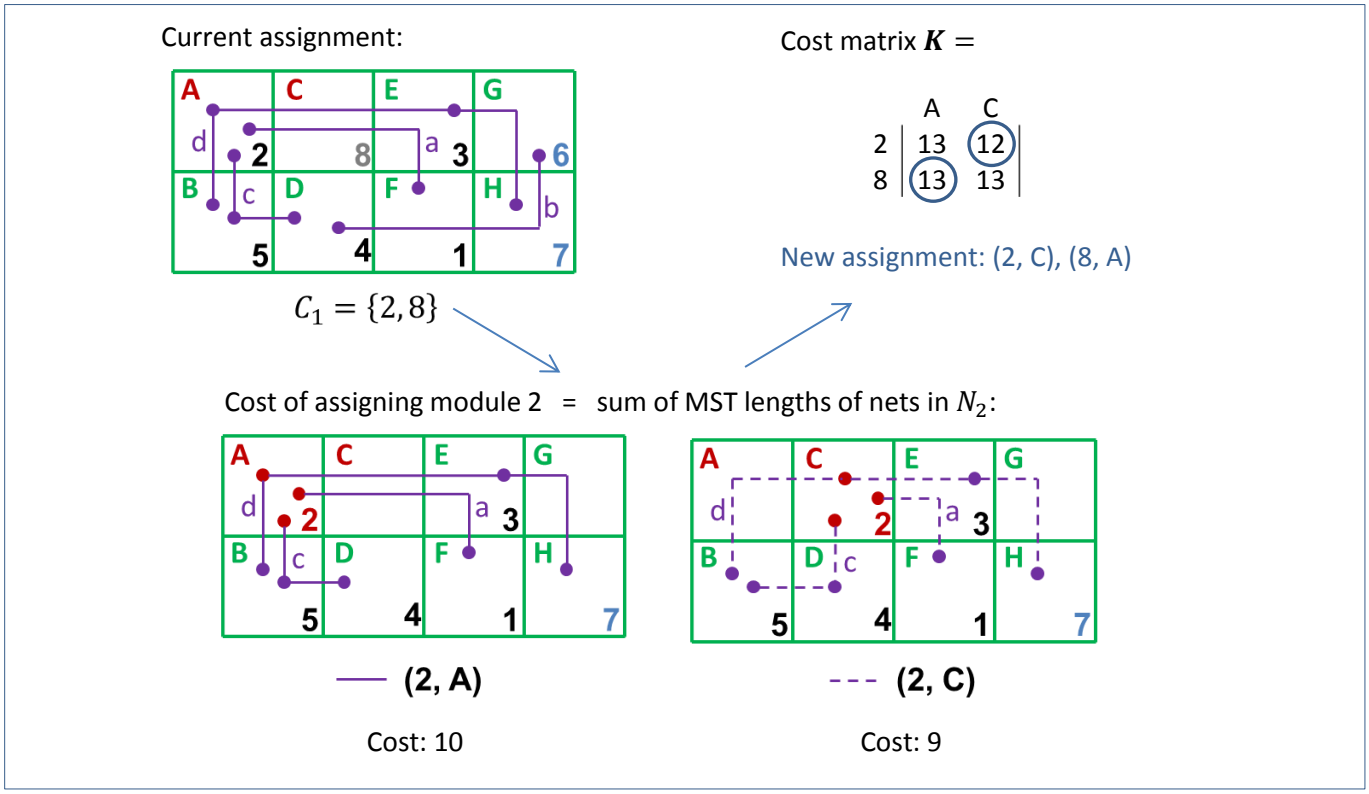
Figure 33: Clique construction of example in Figure 29 according to Theorem 2



4.2 Set up cost matrix and determine new assignment for clique, repeat it for all cliques, then repeat it until no further improvement

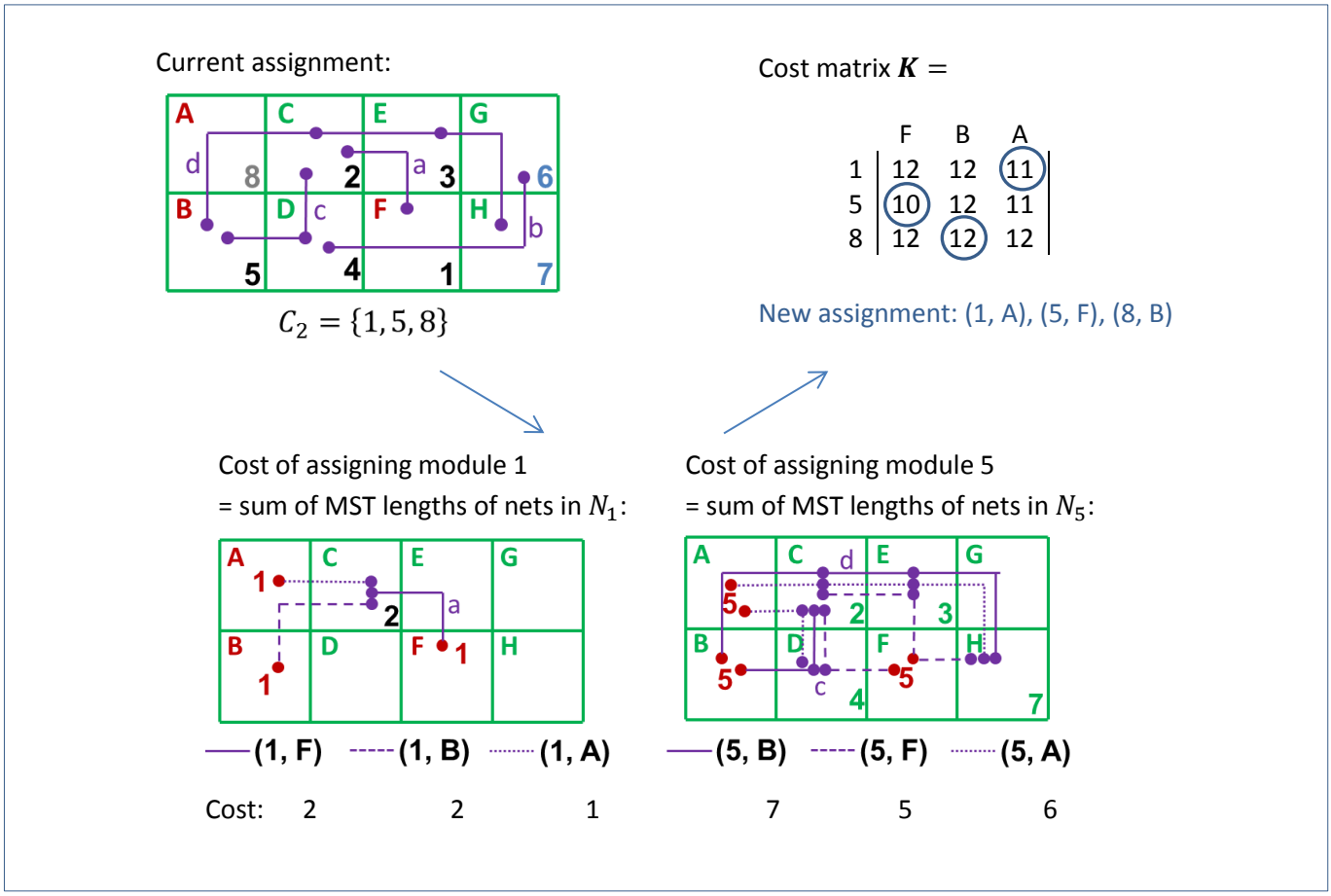
Figure 34 to Figure 37 illustrate the iterative assignment process according to the Steinberg method (Figure 30). In particular, setting up the cost matrix is shown. The netlist example in Figure 29 is taken to illustrate the process. In a first round of three steps, the algorithm goes over the three cliques of independent modules, i.e. modules not connected by a net, as determined in Figure 33. The top left part of Figure 34 shows the initial assignment; the two positions A, C, which correspond to the modules 2 and 8 are marked red. We now investigate the cost of putting 2 and 8 on any of the positions A and C. In the lower part of Figure 34, the cost of putting module 2 on position A (left) and the cost of putting module 2 on position C (right) is determined. Only the nets connected to module 2, i.e., nets a, c, d, are considered. For these nets, the resulting minimum spanning trees and their lengths are determined. Module 2 on position A is the initial situation. We can see that putting module 2 on position C reduces the sum of MST lengths of the affected nets by 1 from 10 to 9. This can be transferred into the overall cost matrix in the upper right part of Figure 34. The cost matrix is ordered such that the current situation is given in the diagonal part: 2 on A, 8 on C, with cost of 13 according to Figure 29. Therefore, the cost of 2 on C is one less, as was determined in the lower part, which results in 12. No computation has to be done for the auxiliary module 8, as it has no connected nets. Inspection of the cost matrix immediately suggests to swap 2 and 8, which puts 2 on C and 8 on A. This gives the new situation in the top left part of Figure 35.

Figure 34: Step 1: cost matrix for clique $C_1 = \{2, 8\}$ of example of Figure 29



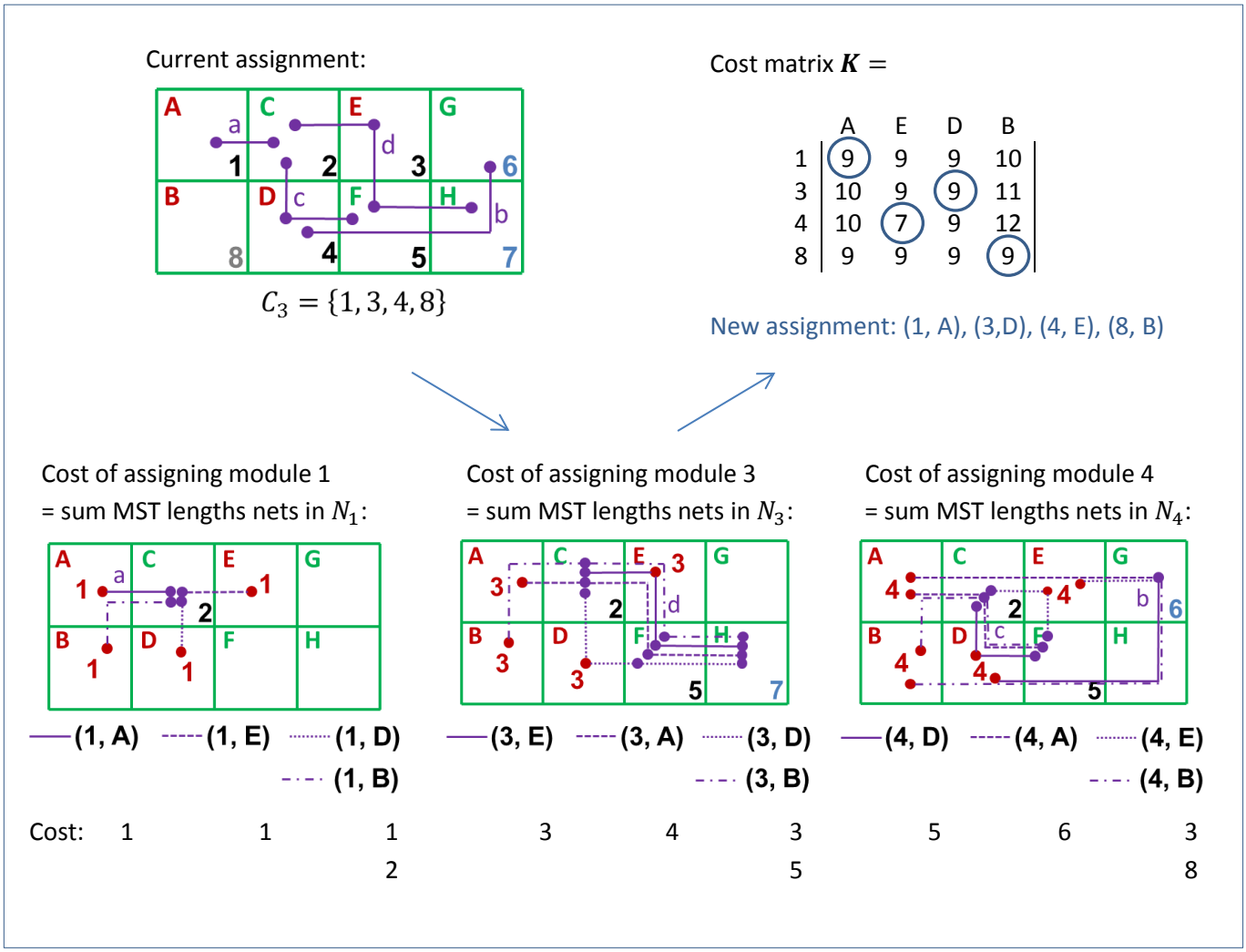
Now, the same procedure is performed for the second clique consisting of modules 1, 5, 8 on positions A, B, F. We investigate the cost of assigning these three modules on these three positions, which is then given on the upper right part of Figure 35. The last line of the cost matrix given there refers to the auxiliary module, which has no nets, the diagonal of the cost matrix refers to the current assignment as before. At this point the overall assignment process, the reason for the partitioning into independent subsets of modules becomes clear: modules 1, 5 (and 8) have no net in common that connects them. Therefore, the assignment of module 1 to one of the available positions A, B, F can be determined before knowing where the modules 5 and 8 will come to lie! The nondiagonal elements in the cost matrix of this subset of modules can be determined independently of each other. The lower part of Figure 35 shows the resulting MSTs of assigning module 1 to positions A, B, F (left) and of assigning module 5 to these positions (right). This gives the resulting MST lengths of nets connected with the modules in the bottom line. With the respective first entry as initial cost, the resulting deltas in cost can be transferred to the cost matrix in the upper right part of the figure. From here the actual mathematical assignment task is starting. It consists in picking the elements in the cost matrix with minimum overall cost under the constraint that every module gets a position and no position gets more than one module. While in Figure 35, this can be done manually as indicated by the blue circles, Section 4.3 will describe an algorithm to solve this problem. The new assignment, (1, A), (5, F), (8, B) gives the starting point for repeating the process for the next independent module set (1, 3, 4, 8) in Figure 36.

Figure 35: Step 2, starting from assignment result of step 1: cost matrix for clique $C_2 = \{1, 5, 8\}$



The third step on the independent module set (1, 3, 4, 8) is illustrated in Figure 36. As before, the cost of assigning the modules 1, 3, 4 on the available positions A, E, D, B is determined through the changes in the MST lengths of the respective connected nets (lower part of the figure) and transferred into the cost matrix in the upper right part of the figure. The blue circles indicate the new assignment (1, A), (3,D), (4, E), (8, B) that results. Its corresponding routing is shown in the top left part of Figure 37.

Figure 36: Step 3, starting from assignment result of step 2: cost matrix for clique $C_3 = \{1, 3, 4, 8\}$

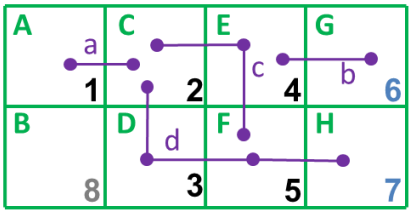


The whole procedure is now repeated, going over all independent module subsets. This repetition happens as long as there is an improvement over a single subset and corresponding revision of the assignment. It only stops, when no further improvement on any subset of independent modules is possible.

The right column of Figure 37 shows the respective cost matrices of the first repetition of this example. It can be seen that no further cost improvement can be achieved on any of these subset. Therefore, the process ends with the final assignment in Figure 37.

Figure 37: Steps 4, 5, 6, starting from assignment result of step 3

Current assignment:



C_1

	C	B
2	(7)	8
8	7	(7)

no improvement!

C_2

	A	F	B
1	(7)	8	8
5	8	(7)	9
8	7	7	(7)

no improvement!

C_3

	A	D	E	B
1	(7)	7	7	8
3	8	(7)	7	9
4	10	9	(7)	12
8	7	7	7	(7)

no improvement!

4.3 The Hungarian method for the mathematical problem of linear assignment

Problem formulation

The task is to find the permutation φ (Eq. 24) with minimum cost $Z(\varphi) = \sum_{i \in M'} k_{i,\varphi(i)}$ (Eq. 28).

$Z(\varphi)$ has following properties:

1. $k_{i,j} \geq 0, i, j = 1, 2, \dots, n_M$, i.e. the cost are always nonnegative. From that follows:
2. $Z(\varphi) \geq 0$ for any permutation φ. And that means:
3. If $Z(\varphi) = 0$, then the corresponding permutation φ is a solution to the linear assignment problem.

We will use the following example of a permutation of four elements to illustrate the method:

Example: $\boldsymbol{K} = \begin{bmatrix} 1 & 2 & 3 & 2 \\ 2 & 5 & 8 & 4 \\ 1 & 4 & 9 & 7 \\ 4 & 8 & 7 & 9 \end{bmatrix}$ $k_{i,j}$ is the cost of assigning module i to position j

An initial permutation φ_S refers to (1, 1), (2, 4), (3, 2), (4, 3). It is obtained by going row(module)-wise and taking the respective minimum available position(column).
It has cost $Z(\varphi_S) = k_{11} + k_{24} + k_{32} + k_{43} = 1 + 4 + 4 + 7 = 16$.

Let us denote the possible permutations with an index and order them such that increasing index refers to increasing cost. Then, φ_1 denotes the permutation with minimum cost:

$Z(\varphi_1) = \min\{Z(\varphi_\lambda) | \lambda = 1, \dots, n_{M'}!\}$

$OZ = \left[Z(\varphi_1) \leq Z(\varphi_2) \leq \cdots \leq Z\left(\varphi_{n_{M'}!}\right)\right]$ (order in cost Z)

$O\varphi = \left[\varphi_1 \preccurlyeq \varphi_2 \preccurlyeq \cdots \preccurlyeq \varphi_{n_{M'}!}\right]$ (order in permutation φ)

In the example, we have:

$OZ = [14 \leq 15 \leq 15 \leq \cdots \leq 24 \leq 24]$

$O\varphi = [\varphi_1 \preccurlyeq \varphi_2 \preccurlyeq \cdots \preccurlyeq \varphi_{24}]$

Solution approach

The idea behind the solution approach is to replace the search for minimum numbers in the cost matrix by a selection of zero elements. This refers to the above mentioned property 3 of the assignment problem: if every selected cost matrix element is zero, then the overall cost is zero and an optimal solution has been achieved To achieve this, a subtraction of values in the matrix is introduced that on the one hand creates zero elements, on the other hand leaves the basic properties of the cost matrix untouched, especially the inherent ordering of permutations according to their cost.

The solution approach hence consists in applying so-called "φ-invariant" operations on cost matrix $\boldsymbol{K}$ until $n_{M'}$ independent zeros are generated in the cost matrix.

A "φ-invariant" operation is an operation that changes the values of the cost matrix without affecting the ordering of the permutations with regard to their cost.

Independence of zeros means that the distribution of zero entries in the transformed cost matrix is such that the minimum cost assignment can be found by adequately selecting zero elements.

There are two basic φ -invariant operations:

1. **Subtraction of a constant value in a row**: The values in row i of cost matrix $\boldsymbol{K}$ determine the order of assigning module i to any available position. If the same constant value is subtracted from all elements in this row, this leaves this order unaltered.
2. **Subtraction of a constant value in a column**: The values in column j of cost matrix $\boldsymbol{K}$ determine the order of assigning any module to position j. If the same constant value is subtracted from all elements in this column, this leaves this order unaltered.

These two operations are applied on a matrix such that zero elements are created. The number of resulting zeros is usually not enough to create an assignment by selection of zeros. Therefore a third operation is introduced, which is a combination of operations 1 and 2 and therefore is also φ –invariant. Operation 3 is repeated until sufficient independent zero elements have been created in the cost matrix.

Operation 1 (Row transformation)

Operation 1 creates zero elements in the rows of the cost matrix:

$$\boldsymbol{K}^{(1)} = \boldsymbol{K} - \begin{bmatrix} u_1 & u_1 & \dots & u_1 \\ u_2 & u_2 & \dots & u_2 \\ & & \vdots & \\ u_{n_{M'}} & u_{n_{M'}} & \dots & u_{n_{M'}} \end{bmatrix}, \quad \text{with } u_i = \min_j u_{ij} \text{ being the smallest element in row } i$$

Operation 1 changes the overall cost of a permutation such that the assignment of module i is decremented by u_i. Summing up decrements of all modules yields the new cost function $Z^{(1)}(\varphi)$ referring to $\boldsymbol{K}^{(1)}$ and any permutation:

$$\forall_\varphi\left[Z^{(1)}(\varphi) = Z(\varphi) - \sum_i u_i\right],\ u_i = \min_j u_{ij}$$

For our example, we obtain:

$$\boldsymbol{K}^{(1)} = \begin{bmatrix}1 & 2 & 3 & 2\\ 2 & 5 & 8 & 4\\ 1 & 4 & 9 & 7\\ 4 & 8 & 7 & 9\end{bmatrix} - \begin{bmatrix}1 & 1 & 1 & 1\\ 2 & 2 & 2 & 2\\ 1 & 1 & 1 & 1\\ 4 & 4 & 4 & 4\end{bmatrix} = \begin{bmatrix}0 & 1 & 2 & 1\\ 0 & 3 & 6 & 2\\ 0 & 3 & 8 & 6\\ 0 & 4 & 3 & 5\end{bmatrix}$$

$$\forall_\varphi\left[Z^{(1)}(\varphi) = Z(\varphi) - (1+2+1+4) = Z(\varphi) - 8\right], \text{e.g.,}\ \ Z^{(1)}(\varphi_S) = 16 - 8 = 8$$

Operation 2 (Column transformation)

Operation 2 creates zero elements in the columns of the cost matrix:

$$\boldsymbol{K}^{(2)} = \boldsymbol{K}^{(1)} - \begin{bmatrix}v_1 & v_2 & \dots & v_{n_{M'}}\\ v_1 & v_2 & \dots & v_{n_{M'}}\\ & & \vdots & \\ v_1 & v_2 & \dots & v_{n_{M'}}\end{bmatrix},\ \text{with}\ v_j = \min_i v_{ij}\ \text{being the smallest element in column } j$$

Operation 2 changes the overall cost of a permutation such that the assignment of any module to position j is decremented by v_i. Summing up decrements of all modules yields the new cost function $Z^{(2)}(\varphi)$ referring to $\boldsymbol{K}^{(2)}$ and any permutation:

$$\forall_\varphi\left[Z^{(2)}(\varphi) = Z^{(1)}(\varphi) - \sum_j v_j\right],\ v_j = \min_i v_{ij}$$

For our example, we obtain:

$$\boldsymbol{K}^{(2)} = \begin{bmatrix}0 & 1 & 2 & 1\\ 0 & 3 & 6 & 2\\ 0 & 3 & 8 & 6\\ 0 & 4 & 3 & 5\end{bmatrix} - \begin{bmatrix}0 & 1 & 2 & 1\\ 0 & 1 & 2 & 1\\ 0 & 1 & 2 & 1\\ 0 & 1 & 2 & 1\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 2 & 4 & 1\\ 0 & 2 & 6 & 5\\ 0 & 3 & 1 & 4\end{bmatrix}$$

$$\forall_\varphi\left[Z^{(2)}(\varphi) = Z^{(1)} - (0+1+2+1) = Z^{(1)}(\varphi) - 4 = Z(\varphi) - 12)\right], \text{e.g.,}\ \ Z^{(2)}(\varphi_S) = 8 - 4 = 16 - 12 = 4$$

Looking at $\boldsymbol{K}^{(2)}$ of the example, it can be seen that although there are 7 zeros, this is not sufficient to get an assignment from a selection of zeros. We could for instance assign module 1 any of the positions 2, 3, 4, then assign any of the modules 2, 3, 4 to position 1. But there is no other zero that tells us how to assign the remaining modules with minimum cost.

In the following, first, operation 3 will be introduced to create more zeros. Then, a theorem will be given that tells us when there are a sufficient number of independent zeros. This theorem gives rise to an algorithm to

determine where to create new zeros in the cost matrix. It is based on a so-called cover line system of column and row lines to cover all existing zeros. A structurally identical algorithm is then given to select the optimal assignment.

Operation 3 (Specific generation of further zeros)

Operation 3 creates further zero elements in the cost matrix by the following combination of operation 1 and operation 2:

1. Determine a cover line system that covers all existing zeros in the cost matrix
2. Determine the minimum value h of all uncovered cost matrix elements.
3. Subtract h in all uncovered rows of the cost matrix (basic φ-invariant operation 1).
4. Add h in all covered columns of the cost matrix (basic φ-invariant operation 2).

The structure of operation 3 can be illustrated as follows for a matrix which as a zero in the first row and a zero in the first column, which are covered by one horizontal and one vertical cover line:

$$\begin{bmatrix} \boxed{0} & & & \\ \boxed{0} & & & \\ & & & \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ h & h & h & h \\ h & h & h & h \\ h & h & h & h \end{bmatrix} + \begin{bmatrix} h & 0 & 0 & 0 \\ h & 0 & 0 & 0 \\ h & 0 & 0 & 0 \\ h & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \boxed{0} & & & \\ \boxed{0} & & & \\ & & & \end{bmatrix} + \begin{bmatrix} h & 0 & & \\ & -h & -h & -h \\ 0 & -h & -h & -h \\ & -h & -h & -h \end{bmatrix}$$

It can be seen that the value h is added on elements that are covered twice, subtracted from uncovered elements; elements covered once remain unchanged:

$$k_{ij}^{(4)} = \begin{cases} k_{ij}^{(2)} - h & \text{if element of } \boldsymbol{K}^{(2)} \text{is uncovered} \\ k_{ij}^{(2)} & \text{if element of } \boldsymbol{K}^{(2)} \text{is covered once} \\ k_{ij}^{(2)} + h & \text{if element of } \boldsymbol{K}^{(2)} \text{is covered twice} \end{cases}$$

The idea behind cover lines is to filter out zeros that refer to possible minimum cost assignments. A horizontal cover line of a zero in the cost matrix means the module that corresponds to this row can be assigned with minimum cost. A vertical cover line of a zero in the cost matrix means that the position that corresponds to this column can be used for a minimum cost assignment. Uncovered elements in the cost matrix refer to modules or positions for which minimum cost assignment decisions cannot be made based on zeros (as there aren't any). Operation 3 as described leads to at least on more zero in this yet "undecided" region and can be repeated until every module and position can be assigned based on a zero in the matrix.

Operation 3 changes the cost of assignments compared to the cost given by $\boldsymbol{K}^{(2)}$. If the number of cover rows is n_{cr}, and the number of cover columns is n_{cc}, we can see that we subtract h in $n_M, - n_{cr}$ rows and add h in n_{cc} rows. This gives:

$\forall_\varphi\left[Z^{(4)}(\varphi) = Z^{(2)}(\varphi) - (n_{M'} - n_{cr}) \cdot h + n_{cc} \cdot h\right]$

$\Leftrightarrow \forall_\varphi\left[Z^{(4)}(\varphi) = Z^{(2)}(\varphi) - (n_{M'} - n_c) \cdot h\right]$, where n_c is the number of cover lines

For our example, we obtain:

$$\boldsymbol{K}^{(4)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 1 \\ 0 & 2 & 6 & 5 \\ 0 & 3 & 1 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 1 & 5 & 4 \\ 0 & 2 & 0 & 3 \end{bmatrix}$$

$\forall_\varphi\left[Z^{(4)}(\varphi) = Z^{(2)}(\varphi) - (4-2) \cdot 1 = Z^{(2)}(\varphi) - 2 = Z(\varphi) - 14\right]$, e.g., $Z^{(4)}(\varphi_S) = 4 - 2 = 16 - 14 = 2$

The question now is if we have enough independent zeros or if we need more operations of type 3. The following theorem helps to get an answer:

Theorem 3 **(König & Egerváry)**

In a matrix with zeros and non-zeros, the minimum number of cover lines of zeros is equal to the maximum number of independent zeros.

Following this theorem, we compute a cover line system for zeros with minimum number of cover lines. If the number of cover lines is equal to the number of modules (= number of positions), then we have enough independent zeros for an optimal assignment. A method to compute a minimum cover line system is described in the following.

Computing a minimum cover line system for matrix zeros

The approach to compute a minimum cover line system of zeros is greedy: in each step, select a row or column with a maximum number of yet uncovered zeros. In this way, a fast coverage of zeros is achieved. Figure 38 shows the corresponding procedure.

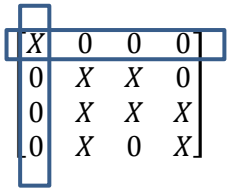
Figure 38: Minimum cover line system according to König & Egerváry

Remaining matrix is initialized with original matrix
Repeat
    For each row i, column j in remaining matrix
        $n_{r,i}$ = number of zeros in row i
        $n_{c,j}$ = number of zeros in column j
    Endfor
    Select row i^* or column j^* with $\max_{i,j}(n_{r,i}, n_{c,j})$ as new cover line
    Delete selected row or column from remaining matrix
Until all zeros covered (i.e., no zero in remaining matrix)

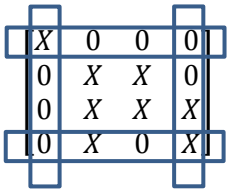
Let us have a look at the structure matrix of the cost matrix $\boldsymbol{K}^{(4)}$ of our example:

$$\begin{bmatrix} X & 0 & 0 & 0 \\ 0 & X & X & 0 \\ 0 & X & X & X \\ 0 & X & 0 & X \end{bmatrix}$$

In the first step, the number of zeros in the 4 rows is determined: 3, 2, 1, 2. The same is done for the columns: 3, 1, 2, 2. We can equally select row 1 or column 1, as this will cover 3 zeros. Let us take row 1, then the column 1, which still allows to cover 3 zeros with on cover line, will be taken in the next step. This yields:



There are two uncovered zeros at this time, which require two more cover lines, e.g.:



Now we have covered all zeros with 4 cover lines. **Theorem** 3 says that we have 4 independent zeros which is equal to the number of modules. Therefore, an optimal assignment can now be done. This brings us to the method to determine an assignment based on independent zeros in the cost matrix, which will be given in the following.

As a side comment, please note that $\boldsymbol{K}^{(4)}$ allows an assignment with zero cost $Z^{(4)}(\varphi_1) = 0$, as it has enough independent zeros. Our initial assignment having $Z^{(4)}(\varphi_S) = 2$ obviously is not optimal.

Computing an assignment on cost matrix with sufficient number of independent zeros

The approach to compute a minimum cover line system of zeros is similar to the minimum cover line computation. The difference is that we prefer to select a zero element that corresponds to a row or a column with only one zero. While the covering computation wants to capture as many zeros as possible in a single step, now we want to keep as much freedom for the further search as possible. From another point of view, we want to take those assignment choices where only on position is optimal for a module (single zero in a row) or where only one module can be assigned to a position at minimum cost (single zero in a column). If rows or columns with single zeros do not exist, we would at least look for a minimum number of zeros. As a result, the algorithm for minimum cover line system computation can be applied, replacing the max by a min operation. Figure 39 shows the corresponding procedure.

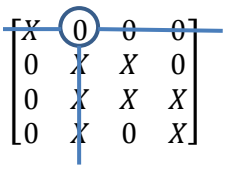
Figure 39: Assignment according to König & Egerváry

Remaining matrix is initialized with original matrix
Repeat
    For each row i, column j in remaining matrix
        $n_{r,i}$ = number of zeros in row i
        $n_{c,j}$ = number of zeros in column j
    Endfor
    Select row i^* or column j^* with $\min_{i,j}(n_{r,i}, n_{c,j})$ as new assignment (i^*, j) or $(i,\ j^*)$, where i or j are row or column number of the corresponding zero
    Delete selected row and column (i^*, j) or $(i,\ j^*)$ from remaining matrix
Until all modules assigned

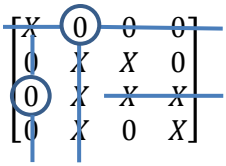
Let us have a look at the structure matrix of the cost matrix $\boldsymbol{K}^{(4)}$ of our example:

$$\begin{bmatrix} X & 0 & 0 & 0 \\ 0 & X & X & 0 \\ 0 & X & X & X \\ 0 & X & 0 & X \end{bmatrix}$$

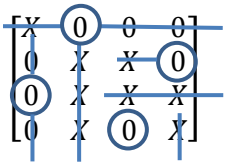
Row 3 and column 2 have only one zero. Let us select column 2, which means we also select row 1 where the single zero in column 2 happens. That in turn means we assign module 1 to position 2. Deleting row 1 and column 2 gives:



Single zeros now are in column 3 or 4 or row 3. Let us select row 3, which means we also select column 1 where the single zero in row 3 happens. That in turn means we assign module 3 to position 1. Deleting row 3 and column 1 gives:



The remaining two zeros uniquely describe the positioning of the remaining two modules:



The corresponding assignment with minimum cost is:

φ_1: $\{(1,2),(2,4),(3,1),(4,3)\}$

$Z^{(4)}(\varphi_1) = 0\,; Z(\varphi_1) = Z^{(4)}(\varphi_1) + 14 = 14$

Rectangular cost matrix

The general assignment problem includes the case of rectangular cost matrices. These are transformed into quadratic cost matrices by adding rows or columns with zeros as required:

$$\boldsymbol{K}_{<m\,x\,n>}: \begin{cases} \text{if} & m < n & \text{insert } n-m \text{ zero rows} \rightarrow \text{no operation 2 (each column has a zero)} \\ \text{if} & m > n & \text{insert } m-n \text{ zero columns} \rightarrow \text{no operation 1 (each row has a zero)} \end{cases}$$

Overview of the Hungarian method

This is the overview of the method described in this Section 4.3:

Figure 40: Overview of the Hungarian method for linear assignment

If cost matric rectangular *then*
    Make cost matrix quadratic
endif
Perform operation 1
Perform operation 2
Repeat
    Compute minimum cover line system for zeros
    If not enough independent zeros *then*
        Perform operation 3
    endif
Until enough independent zeros
Compute assignment

5. Quadratic placement

Chapter 4 dealt with a method for assignment (Section 3.1), where modules are assigned to given positions. Specifically, a method for linear assignment on subsets of independent modules (Section 3.5) was described.

This chapter will present an approach for arrangement (Section 3.2), where there are no given positions, and the modules can be freely arranged on a given area. The widely used approach is quadratic placement, which refers to the clique net model (Section 3.4).

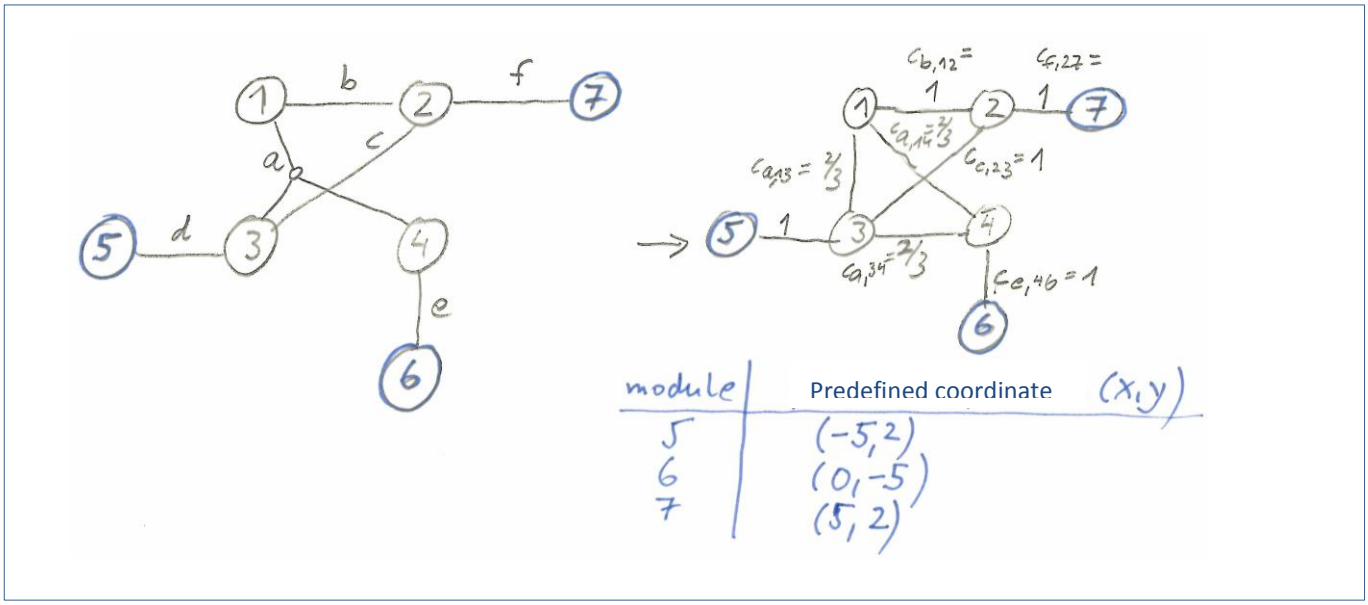
We will first give a heuristic approach to the formulation of quadratic placement. After that, analogies to mechanics and electrical networks will be pointed out. The mechanics analogy will make clear why quadratic placement is also denoted as **force-directed placement**.

An example problem to illustrate quadratic placement is given in the following.

Introducing an example to illustrate quadratic placement

Figure 41 shows an example with 7 modules and 6 nets. The modules 5 to 7 have predefined coordinates. The n-th order nets on the left side are transformed into cliques of two pin nets on the right side according to Section 3.4.

Figure 41: Quadratic placement example



If we allow the multigraph representation, then every net v creates a complete subgraph with undirected edges among the modules in the module set M_v of net v. Each of these edges has weight $2/|M_v|$ according to Section 3.4. The edges are denoted by v, ij and collected in the set S:

Eq. 35 $$S = \{ v, ij \mid v \in N \ \wedge \ i, j \in M_v \ \wedge \ j > i \}$$
Edge v, ij has weight $c_{v,ij} = \frac{2}{|M_v|}$

In the example in Figure 41, we have $S = \{a, 13;\ a, 14;\ a, 34;\ b, 12;\ c, 23;\ d, 35;\ e, 46;\ f, 27\}$ with the given weights.

5.1 Heuristic formulation of quadratic placement

The heuristic approach models the net length contribution of an edge in the circuit graph (e.g., Figure 41 right) through the quadratic distance of the modules connected by this edge:

$$L_{v,ij}^{(xy)} = \frac{1}{2} c_{v,ij} \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)$$

The overall net length is then obtained by summing up the contributions from all edges in S:

Eq. 36 $$L^{(xy)}(x_1, \dots, x_M, y_1, \dots, y_M) = \underbrace{\sum_{v,ij \in S} \frac{1}{2} c_{v,ij} (x_i - x_j)^2}_{L^{(x)}(x_1, \dots, x_M)} + \underbrace{\sum_{v,ij \in S} \frac{1}{2} c_{v,ij} (y_i - y_j)^2}_{L^{(y)}(y_1, \dots, y_M)}$$

We can see that the x- and y-coordinates can be separated into independent parts of the objective function. That means that the solution in x- and y-coordinate can be computed independently, and in parallel.

In the following the description will be done for the x-coordinates. Y-coordinates hold analogously.

Eq. 37 $$\min_{x_1, \dots, x_M} L(x_1, \dots, x_M) \quad \text{with} \quad L(x_1, \dots, x_M) = \sum_{v,ij \in S} \frac{1}{2} c_{v,ij} (x_i - x_j)^2$$

We will take a closer look at the structure of Eq. 37 with regard to modules with predefined coordinates and modules whose coordinates are to be computed.

The set of edges in the two-vertices clique circuit graph can be partitioned into three disjunct subsets:

Eq. 38
S_0: edges v, ij where *neither* i nor j has predefined coordinates
S_j: edges v, ij where *only* j has predefined coordinates
S_{ij}: edges v, ij where *both* i and j have predefined coordinates
$$S = S_0 \cup S_j \cup S_{ij} \ ; \ S_0 \cap S_j = S_0 \cap S_{ij} = S_j \cap S_{ij} = \{\}$$

$L\ (x_1, \dots, x_M)$ in Eq. 37 can be partitioned according to these three sets:

Eq. 39 $$L\ (x_1, \dots, x_M) = \sum_{v,ij \in S_0} \frac{1}{2} c_{v,ij} (x_i - x_j)^2 + \sum_{v,ij \in S_j} \left[\frac{1}{2} c_{v,ij} (x_i)^2 - (c_{v,ij} x_j) x_i + \frac{1}{2} c_{v,ij} (x_j)^2 \right] + + \sum_{v,ij \in S_{ij}} \frac{1}{2} c_{v,ij} (x_i - x_j)^2$$

In Eq. 39 the coordinates that refer to predefined values are marked in red. They are constants. The remaining variables refer to modules to be placed, $M_\square$, and are marked in green. It can be seen that Eq. 39 refers to following structure in matrix/vector notation:

Eq. 40 $$L\ (\boldsymbol{x}_\square) = \frac{1}{2} \boldsymbol{x}_\square^T \cdot \boldsymbol{C} \cdot \boldsymbol{x}_\square + \boldsymbol{d}^T \cdot \boldsymbol{x}_\square + const$$
where $\boldsymbol{x}_\square$ denotes the coordinates of modules to be placed

The linear terms stem from the edge set S_j, where one module has fixed coordinates, as in the second sum of Eq. 39, second term.

At this point, we have not explained how the matrix $\boldsymbol{C}$ and vector $\boldsymbol{d}^T$ are actually set up. This will be done in Section 5.4. For the moment, we just want to conclude from the fact that we have a quadratic objective, as Eq. 40 features, hence the term quadratic placement:

Eq. 41 $$\min_{\boldsymbol{x}_\square} L\ (\boldsymbol{x}_\square), \quad \text{with } L\ (\boldsymbol{x}_\square) = \frac{1}{2} \boldsymbol{x}_\square^T \cdot \boldsymbol{C} \cdot \boldsymbol{x}_\square + \boldsymbol{d}^T \cdot \boldsymbol{x}_\square + const$$

The solution is obtained by setting the first derivative of the objective function equal to zero, which yields a linear equation system ($\boldsymbol{C}$ symmetric):

Eq. 42 $$\nabla L\ (\boldsymbol{x}_\square) = \boldsymbol{0}: \quad \boldsymbol{C} \cdot \boldsymbol{x}_\square + \boldsymbol{d} \quad = \boldsymbol{0}$$

The second order optimality condition requires that that second order derivative of the objective function is positive definite:

Eq. 43 $\nabla^2 L\ (\boldsymbol{x}_\square)$ positive definite: $\boldsymbol{C}$ positive definite

The symmetry and positive definiteness of $\boldsymbol{C}$ can be deduced from its setup.

Quadratic placement according Eq. 42 hence consists in the solution of a linear equations system. This equation system is usually very large (e.g., millions of coordinates) and sparse, suitable methods to solve it are, e.g., conjugate gradient based approaches.

Please note that the term $\boldsymbol{d}_x$ in Eq. 42 stems from the existence of modules with predefined coordinates. If there is no module with predefined coordinates, then $\boldsymbol{d}_x = \boldsymbol{0}$, and the optimal solution would be the trivial solution $\boldsymbol{x}_\square^* = \boldsymbol{0}$. This is clear from the objective of minimum distance, which is achieved if all modules sit on one another. Therefore, quadratic placement requires measures to prevent this trivial solution. One (more important will be mentioned later) ingredient here are fixed modules at the borders of the placement area, e.g., microchip pad positions.

5.2 Mechanics approach: force-directed placement

The mechanics approach interprets the routing length/cost of an edge $v, ij \in S$ through a spring between the two modules i, j:

Eq. 44 **cost of edge** $v, ij \in S$ $\sim$ **energy of spring between modules** i, j **with spring rate** $c_{v,ij}$

The analogy can be confirmed by looking at the respective equations from mechanics. For the force holds:

Eq. 45 $$\underbrace{F}_{\text{force}} = \underbrace{D}_{\substack{\text{spring}\\ \text{rate}}} \cdot \left(\underbrace{x - x_0}_{\text{deflection}} \right)$$

Then we get for the energy:

Eq. 46 $$\underbrace{E}_{\text{energy}} = \int F \cdot dx = \frac{D}{2} \cdot (x - x_0)^2 \quad \rightarrow \quad \frac{1}{2} c_{v,ij} (x_i - x_j)^2$$

This is the same form as our heuristic approach. The proximity requirement due to routing length is in this way modeled by a spring that pulls two modules together. Hence the name “force-directed placement”.

The optimization problem now consists in obtaining an **equilibrium with minimum overall energy**, which is equivalent to obtaining an **equilibrium of forces** for the **overall system** as well as for **each module**:

Eq. 47 $$\boldsymbol{F} = \nabla L \left(\boldsymbol{x}_{\square} \right) = \boldsymbol{C} \cdot \boldsymbol{x}_{\square} + \boldsymbol{d} \quad = \boldsymbol{0}$$

$$\bigvee_i F_i = \nabla L \left(\boldsymbol{x}_{\square,i} \right) = 0$$

In literature, the term “force-directed placement” is used more often than “quadratic placement”. This is due to the fact that additional forces are introduced to solve the problem of module lumps. It was described in preceding section that all modules are on top of each other if there is no predefined module position. Even if there are pads or some modules with predefined coordinates, the many remaining modules will tend to placed too close to each other in the original problem formulation, because too many springs are there that pull them together. Therefore, additional forces are added, such as forces that pull modules to available free space. These additional forces are consistent with the original approach of an optimization problem with a quadratic objective whose solution is obtained through a linear equation system, yet other physical interpretations are used to approach the solution. But even with additional forces, the quadratic placement will end with overlapping modules. To finalize the placement to non-overlapping modules, another step called **legalization** is required.

5.3 Electrical network approach: DC analysis

There is another interpretation of quadratic placement, which is from network theory. Here we have the following analogies:

Module coordinates ≡ node voltages

Edge weights ≡ branch admittances

Predefined module coordinate of module ≡ voltage source at respective node

The placement optimization problem now is formulated as **Kirchhoff current law at circuit nodes**, which refers to a DC analysis, whose solution is a **DC operating point**.

This approach or interpretation, respectively, can be derived if we look at the rules to set up the linear equation system for quadratic placement, as done in the following section.

5.4 Setting up the linear equation system of quadratic placement

We can distinguish between two cases when we set up the equation system for quadratic placement (Eq. 42)

- Both modules are variable (Figure 42).
- One module is variable, one module is fixed (Figure 43).

Figure 42 and Figure 43 illustrate these two cases with respective edges from the example in Figure 41. The corresponding cost function is given, after that the derivatives are computed and inserted into the overall equation system. We recognize that the setup rules are identical to those of the equation system of a **nodal analysis** and we can confirm the analogies given in the preceding Section 5.3.

The overall **equivalent circuit** to the quadratic placement problem can be set up by making:

- **edge to admittance with the edge weight as value,**
- **fixed coordinate to a current source with edge weight times coordinate as value**.

Then, the equation system is set up as follows:

Eq. 48 $\boldsymbol{C}$: C_{ii} = **sum of admittances** $c_{v,ij}$ **at node** i

$C_{ij} = C_{ji}$ = **negative admittance** $c_{v,ij}$ **between node** i **and node** j

$\boldsymbol{d}$: d_i = **sum of current sources at node** i **(arriving at node:** $+$**, leaving node:** $-$ **)**

Figure 42: "Stamp" of an edge with two variable modules in Eq. 42

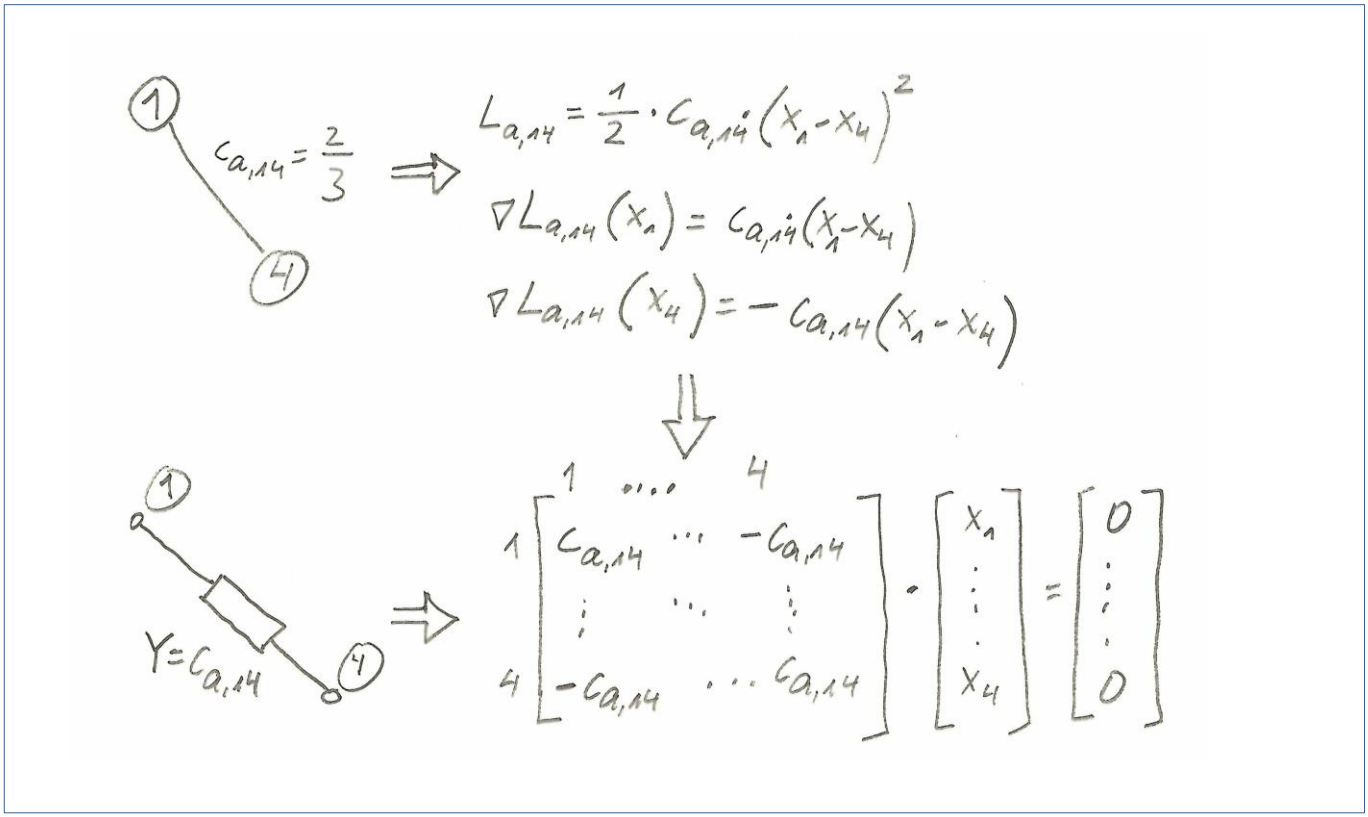
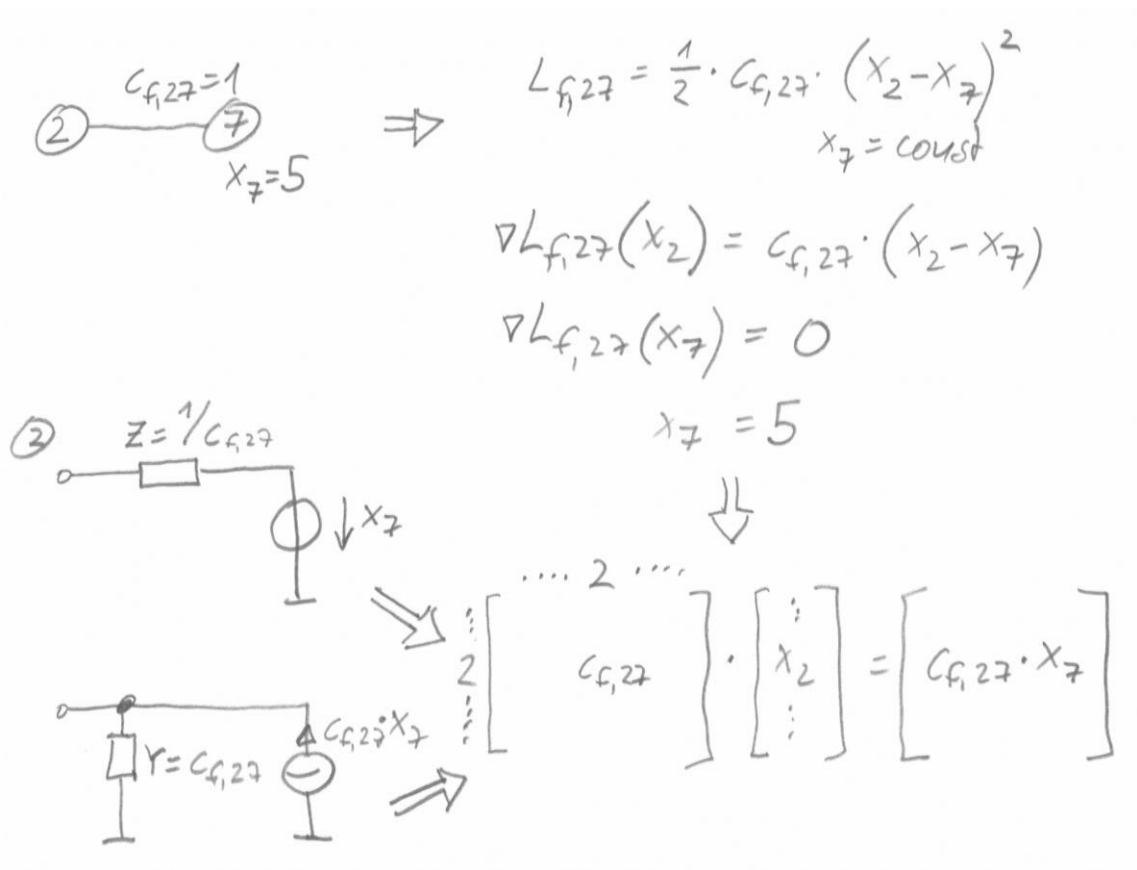
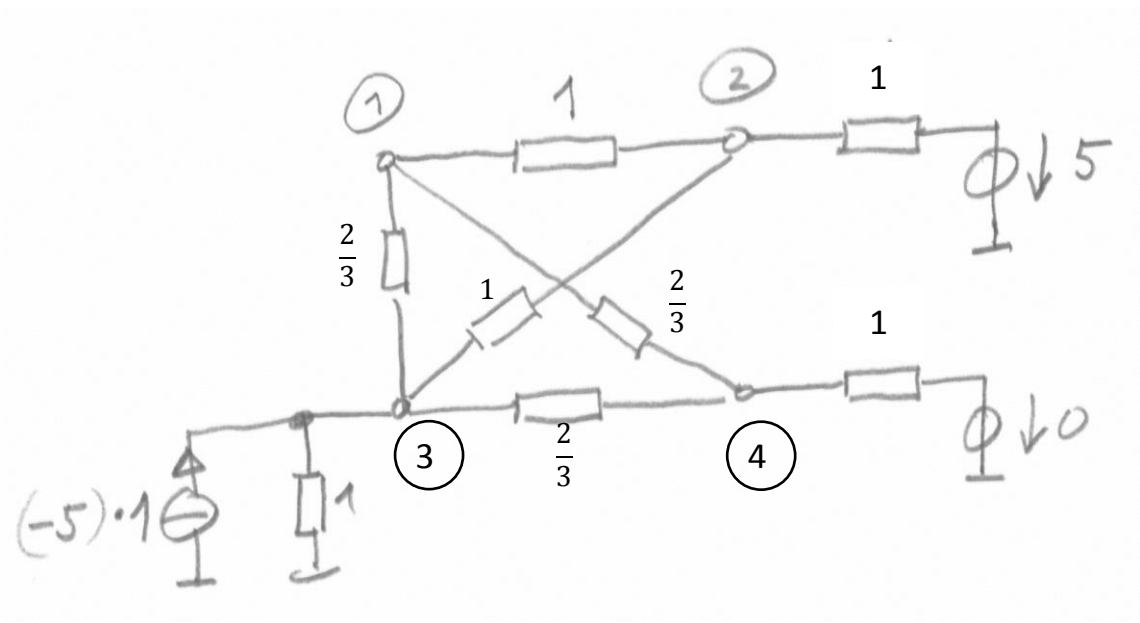


Figure 43: "Stamp" of an edge with one variable and one fixed module in Eq. 42



The overall equivalent circuit of the example in Figure 41 thus becomes:

Figure 44: "Stamp" of an edge with one variable and one fixed module in Eq. 42



The corresponding equation system is:

$$\begin{bmatrix} 1+\frac{2}{3}+\frac{2}{3} & -1 & -\frac{2}{3} & -\frac{2}{3} \\ -1 & 1+1+1 & -1 & 0 \\ -\frac{2}{3} & -1 & 1+1+\frac{2}{3}+\frac{2}{3} & -\frac{2}{3} \\ -\frac{2}{3} & 0 & -\frac{2}{3} & 1+\frac{2}{3}+\frac{2}{3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \cdot 5 \\ 1 \cdot (-5) \\ 1 \cdot 0 \end{bmatrix}$$

It has the solution:

$$\begin{bmatrix} x_1^* \\ x_2^* \\ x_3^* \\ x_4^* \end{bmatrix} = \begin{bmatrix} 0.1948 \\ 1.3636 \\ -1.1039 \\ -0.2597 \end{bmatrix}$$

6. Simulated Annealing

Simulated annealing is, like Monte Carlo analysis, a stochastic optimization procedure with a simple basic structure. It mimics the physical cooling of liquid metal to its crystal structure. The structure of the algorithm is given in Figure 45. The core of simulated annealing is that

- new solutions are created randomly with a distance to the previous solution that is directly related to the temperature,
- worse solutions are accepted with a certain probability, which is decreasing with temperature and cost.

The algorithm is continuously decreasing the temperature, such that the current solution can widely vary between different cost values and among the potential solutions in the beginning and less and less in the course of the simulated time.

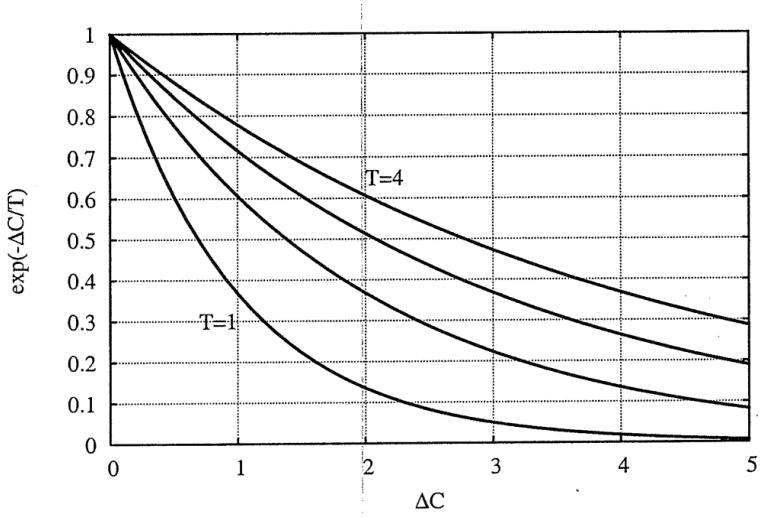
Figure 46 shows examples of the random selection of worse solutions.

Figure 45: Simulated annealing

$S \coloneqq S_0$ // find an initial solution
$T \coloneqq T_0$ // set an initial temperature
$\alpha \coloneqq \alpha_0 < 1$ // set the cooling rate
$\tau \coloneqq \tau_0$ // set an initial time interval
$\beta \coloneqq \beta_0 > 1$ // set the slowing rate for time interval
$t \coloneqq 0$ // set the initial time

Repeat
  For $i = \tau, 1, -1$ // repeat the following τ times
    Create S_{trial} with random direction $\frac{S_{\text{trial}} - S}{\|S_{\text{trial}} - S\|}$, and length $\|S_{\text{trial}} - S\| \sim T$ or $\sqrt{T}$
    $\Delta C \coloneqq \text{cost}(S_{\text{trial}}) - \text{cost}(S)$ // difference in cost between previous and trial solution
    If $\Delta C < 0$ ***or*** $\text{uniformrandom} < e^{-\Delta C / T}$ // trial solution taken randomly or if better
      $S \coloneqq S_{\text{trial}}$
    Endif
  Endfor
  $t \coloneqq t + \tau$ // increase the current time
  $T \coloneqq \propto \cdot T$ // reduce temperature by given cooling rate
  $\tau \coloneqq \beta \cdot \tau$ // increase time interval by given slowing rate
until $\tau \geq \tau_{\max}$ // stop if maximum time is reached

Figure 46: New solution is accepted for a (0,1) uniform random number below a respective curve



7. Grid routing, maze routing, LEE algorithm, DIJKSTRA algorithm

With this chapter, we move on to routing of integrated circuits. A basic approach to routing is the so-called grid routing, which finds the shortest path between two points (i.e., pins) over a discrete grid of possible steps on a routing area that includes obstacle regions. The solution is based on the LEE algorithm, which is a problem-specific form of the general shortest path problem. The general shortest path problem is solved by Dijkstra's algorithm, which will be described in the following.

7.1 Shortest path algorithm of DIJKSTRA

Dijkstra's algorithm computes all paths from a vertex a to all other vertices in a weighted root graph $G = (V_v, E_v, d_{v,ij})$. A **root graph/root tree** is defined as a directed graph with a distinct root vertex, from which paths to all other vertices in the graph exist.

Dijkstra's algorithm is structurally identical to Prim's algorithm (Figure 24, Section 2.4, page 25). The differences are:

- Dijkstra works on directed graph.
- With the merger of edges the weights are updated to denote the path lengths to the respective successor nodes.

The algorithm is sketched in Figure 47.

Figure 47: Shortest path algorithm of Dijkstra

Initially, $V_{SP} := \{a\}$ (root vertex), $E_{SP} := \{\ \}$, $\forall_{n \in \Gamma^+(a)} \text{label}(n) := a$, $V \coloneqq V_0$, $E \coloneqq E_0$, $d_{ij} = d_{0,ij}$

Repeat

Select successor vertex z of vertex a with minimum edge weight d_{az} (sorting algorithm):

$$(a,z) = \text{argmin}_{(a,n)} \left\{ d_{an} \middle| n \in \Gamma^+(a) \right\}$$

"Shortest path graph construction":

Add new vertex and edge to G_{SP}:

$V_{SP} \cup = \{z\}$, $E_{SP} \cup = \{(\text{label}(z), z)\}$

(label(z) denotes the predecessor of z that corresponds to the minimum edge weight d_{az})

New shortest path:

$P_{\min}(a,z) = P_{\min}(a, \text{label}(z)) \cup (\text{label}(z), z)$, $L_{\text{Pmin}}(a,z) = d_{\text{az}}$; $\text{label}(n) := z$

"Edge revision for merger of a, z":

For all successors n of vertex z except a : $n \in \Gamma^+(a) \setminus \{a\}$

If edge (a,n) did not exist

Create edge (a,n): $E \coloneqq E \cup \{(a,n)\}$; $d_{an} \coloneqq d_{az} + d_{zn}$

Else (check if new edge from a to n induced by z has lower weight than old edge)

If $d_{az} + d_{zn} < d_{an}$

Update edge weight and label: $d_{an} \coloneqq d_{az} + d_{zn}$; $\text{label}(n) := z$

Endif

Endif

Endfor

"Problem reduction":

Delete "swallowed" vertex z and its edges to predecessors and to successors from graph:

$E := E \setminus \{ (z,n) \mid n \in \Gamma^+(a) \} \setminus \{ (n,z) \mid n \in \Gamma^-(a) \}$, $V = V \setminus \{z\}$

Until $|E_{SP}| = |V_0| - 1$, i.e., the number of edges in the shortest paths graph is one less than the number of vertices in the graph

An example that illustrates the steps of the Dijkstra algorithm is given in Figure 48 and Figure 49.

In each step,

- the respective shortest path graph is denoted and drawn in green, where $a^{(i)}$ stands for the "super" node as a black box and $G_{SP}^{(i)}$ stands for the graph as a white box,
- the priority queue and the resulting next shortest path and shortest path length are given,
- new edges and new edge weights are drawn in red.

The starting point is node a of the original graph, we assume that we have found the root node at this point. The successor nodes are b, x, c who are labeled accordingly. We pick the minimum path length, which refers to the edge to node b. The respective graph given in the second column is obtained. The new super node $a^{(1)}$ gets two new edges to nodes y, z, respectively, which are marked in red. The corresponding edge weights are obtained by adding the weight of edge (a, b) to the edge (b, y) and (b, z), respectively. This yields the values 9 and 11, which tell us the path lengths from a to y, z, respectively. An edge from a to node x existed, but we get a new path from a over b to x with length 8, which is shorter than the existing path, which goes directly from a to x with length 9. Therefore, the label of node x, which denotes its predecessor in node $a^{(1)}$ is changed to b, and the edge weight of $(a^{(1)}, x)$ is updated to 8. The whole process is repeated. Interestingly, only the last step (Figure 49) reveals the shortest path to node z, which goes over node y.

Figure 48: Example process of shortest path construction with the Dijkstra method

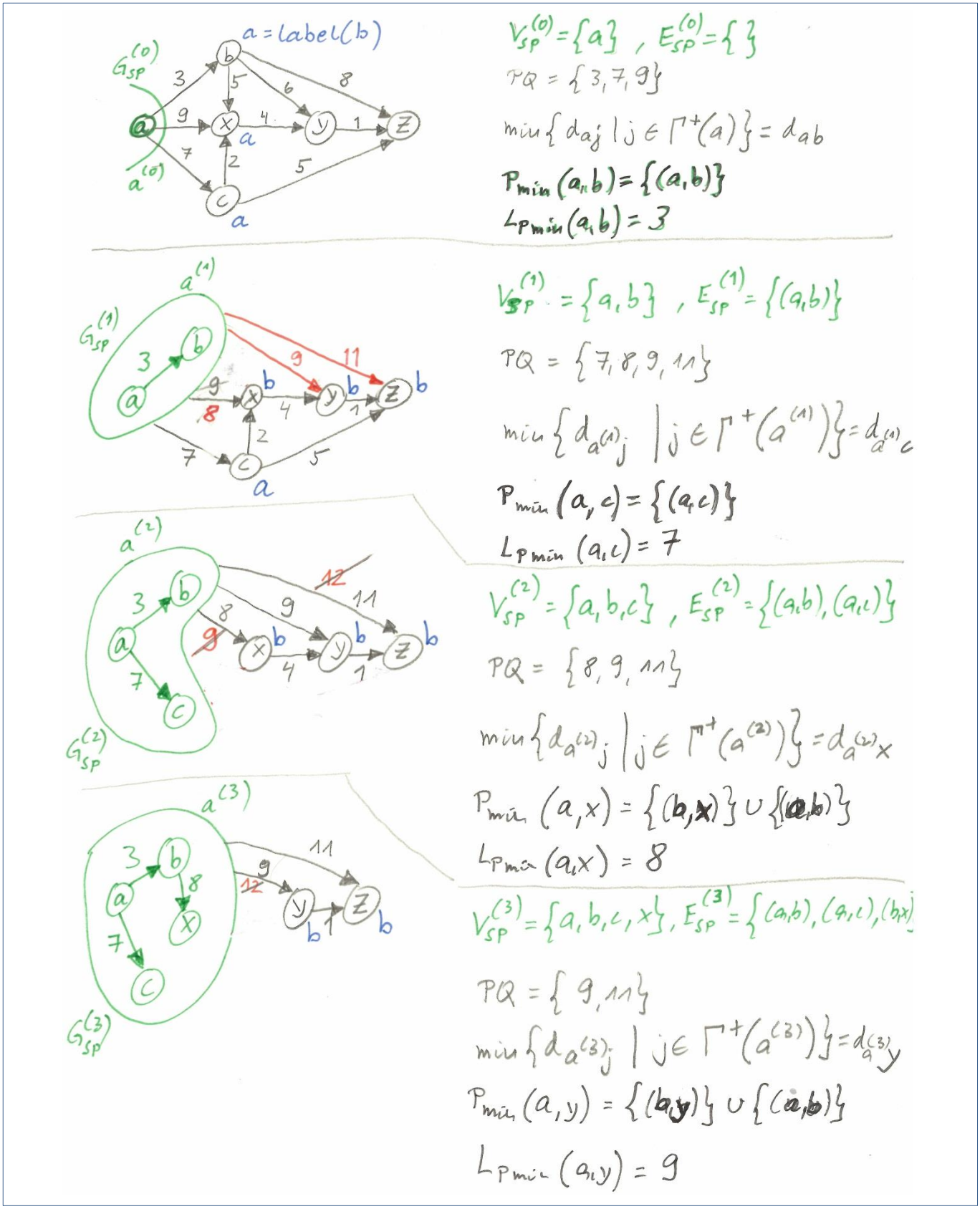
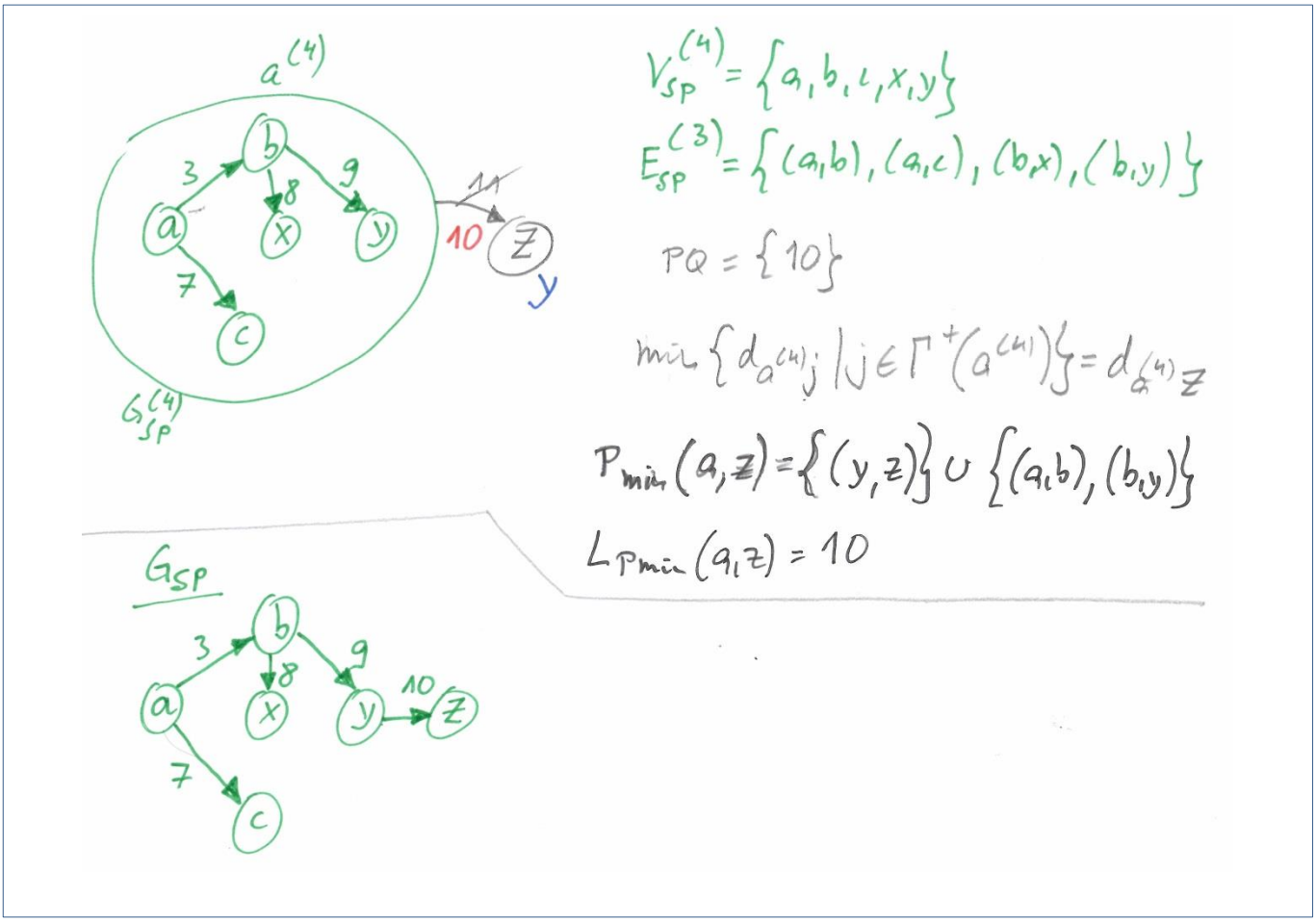


Figure 49: Cont'd: example process of shortest path construction with the Dijkstra method



7.2 LEE algorithm for maze routing

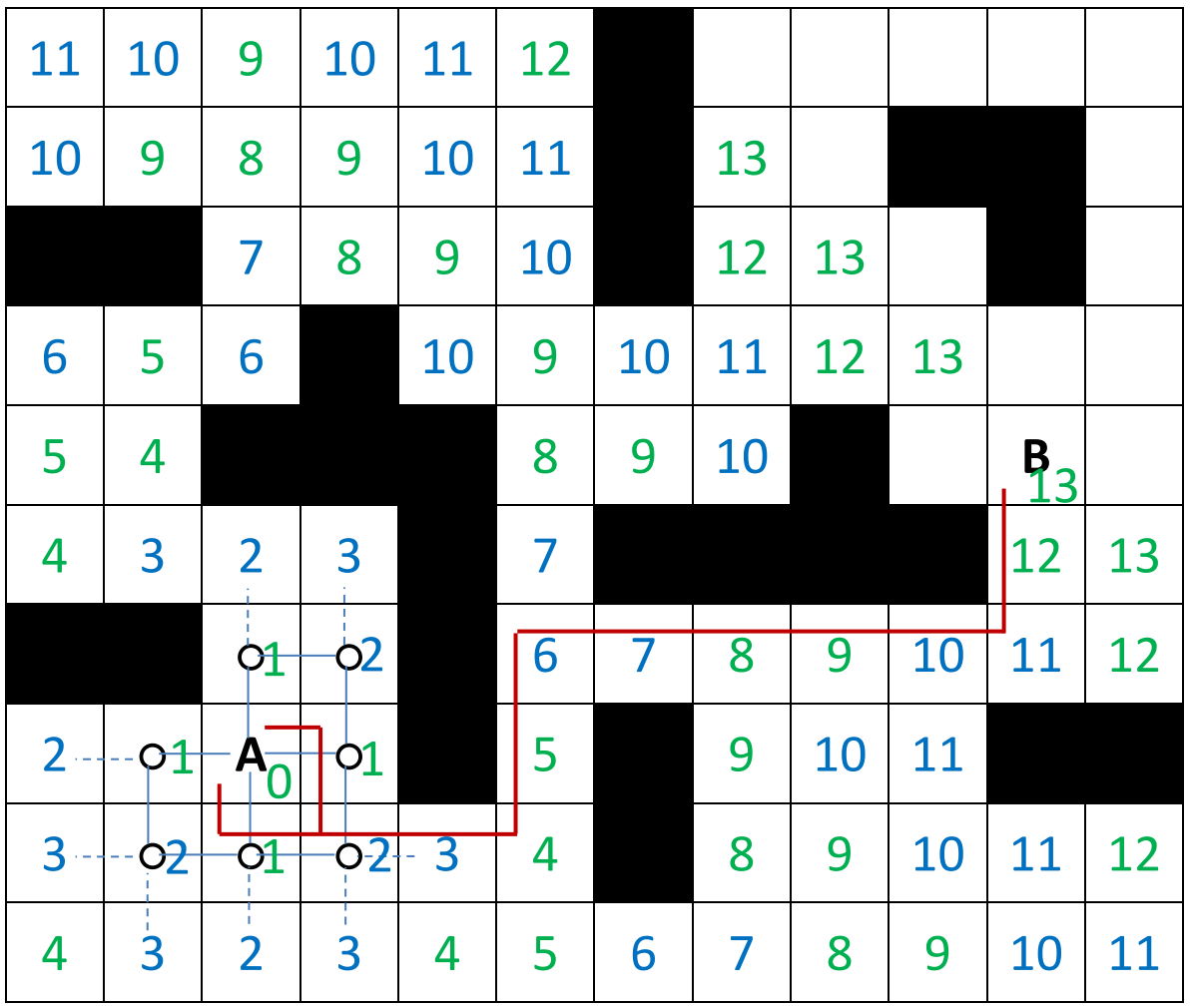
The basic routing problem consists in determining a connection topology of the wiring between the pins of a circuit, after the placement of the modules has been determined.

Objectives of routing are, e.g., minimum wire length, minimum number of vias (i.e., switching between routing layers), minimum crosstalk.

Usually the available routing area is discretized into small area elements. A wire consists of moves from one element to another. Usually, jumps in vertical or horizontal direction are permitted, diagonal moves are excluded.

Figure 50 illustrates a routing area grid with 10 by 12 grid points and certain obstacles that form a “maze”.

Figure 50: Example of LEE algorithm for maze routing



Every area element is represented as a vertex (“grid point”) of a graph, edges describe available moves between grid points. In Figure 50, vertices and edges of the resulting grid graph are illustrated in the surrounding of some pin “A”. Pin “A” has to be connected to pin “B” with shortest wire length. This translates into a shortest path problem. The grid graph has specific properties:

- The graph has mesh structure. In 2D, each grid point has maximum 4 neighbors.
- Each edge has unit weight 1.
- Edges are undirected.

The specific structure of the problem enables a specific version of Dijkstra’s algorithm, which is the LEE algorithm. It basically consists of:

- Choose start vertex.
- Propagate forward, increment by 1 for each move along grid graph, until target vertex has been reached.
- Backtrack from target vertex to start vertex along decreasing grid numbering.

Figure 50 illustrates the process starting from “A” till pin “B” is reached with a path length of 13. Backtracking from “B” results in the red path. Around “A”, there are two alternatives with same path length, where one has a smaller number of bends and may therefore be preferred.

Please note that directed edges evolve dynamically during the LEE algorithm.

If we think of a 20mm x 20mm chip with a grid size of 200 nm, then we obtain 10^{10} gridpoints. A route from one corner to the other on the diagonal side would be 40 mm, which refers to 200,000 gridpoints. To store this number, we need 18 bits. If we allocate 18 bits for every gridpoint, we would need 180 GB, which is ridiculous.

Instead, we can use a so-called **Akers coding**:

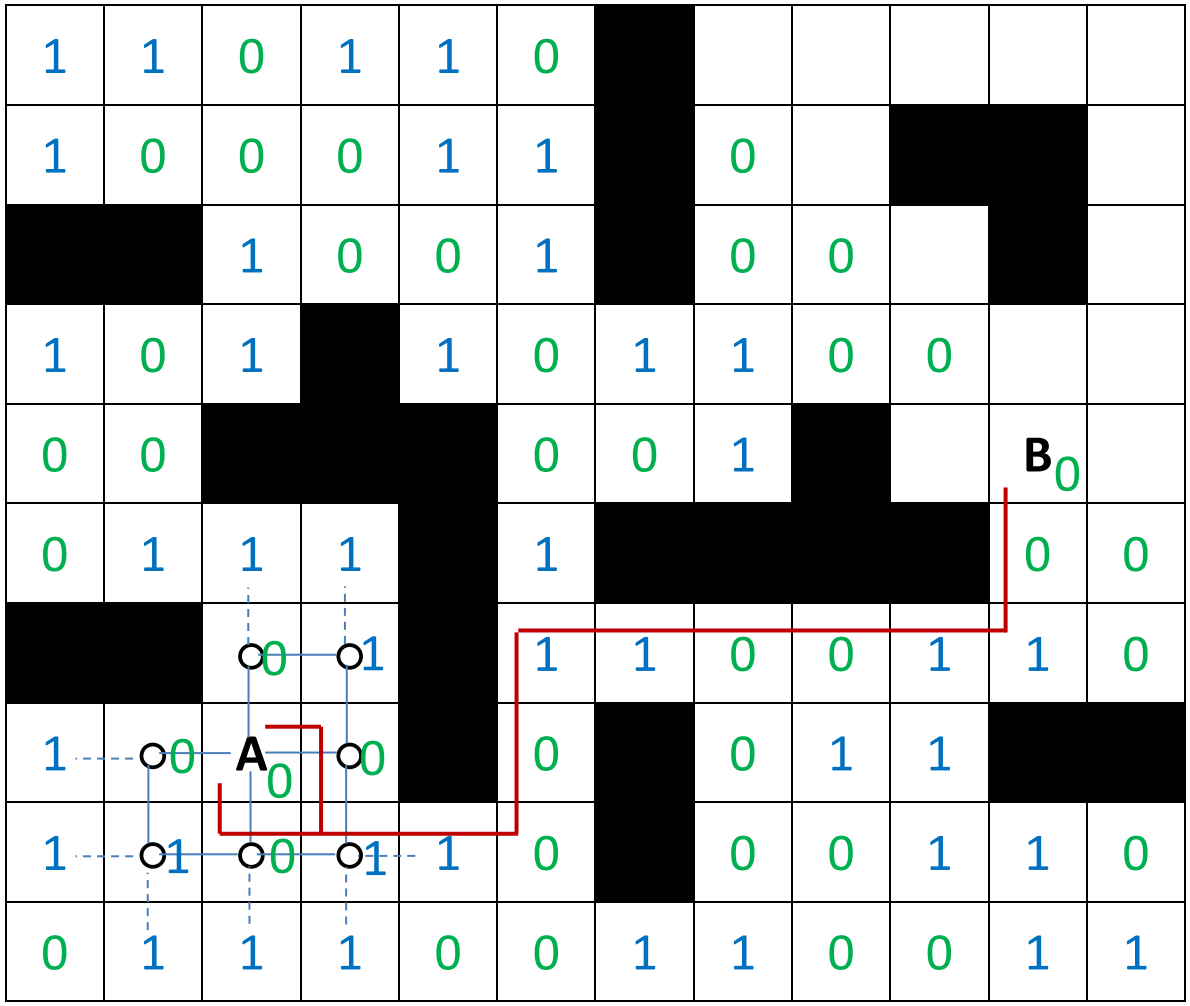
Increment	1	2	3	4	5	6	7	8	…
Akers	0	0	1	1	0	0	1	1	…

It alternates between two subsequent 0’s and two subsequent 1’s and allows a unique backtracking.

Figure 51 shows the previous example with Akers coding.

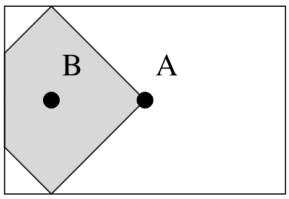
Please note that the Akers coding means that the numbers during the path propagation do not contain the path length, they are to allow backtracking. The path length is computed during backtracking by incrementing the corresponding variable.

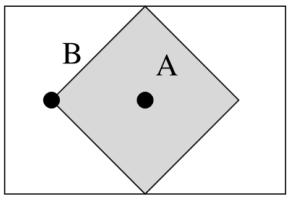
Figure 51: Example of LEE algorithm with Akers coding



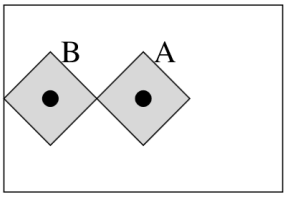
Modifications of the LEE algorithm can be done, for instance, with regard to:

- Choice of starting point

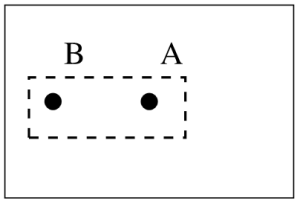




- Simultaneous propagation from starting point and target point



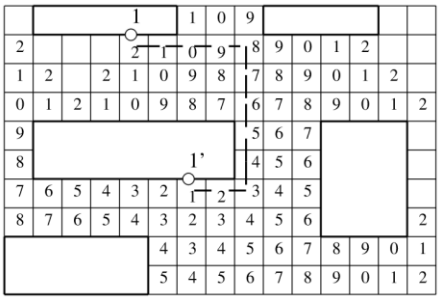
- Search windows



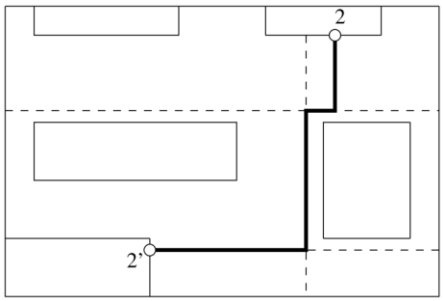
8. Routing methods

There are several basic approaches to routing, as for instance:

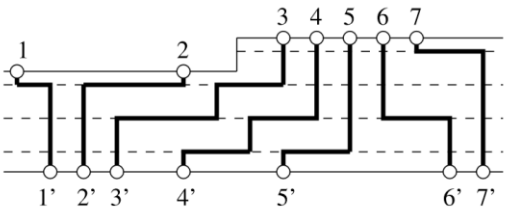
- Grid routing (see previous section 7, page 67)



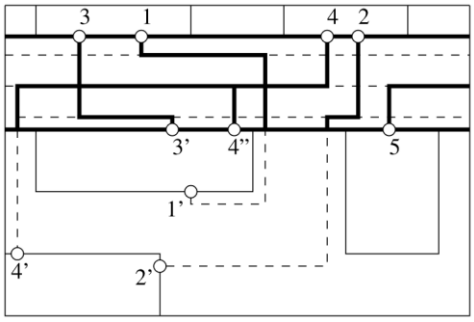
- Line routing (line propagation in horizontal and alternatingly vertical direction)



- River routing (single layer connections between upper and lower riverbank, assignment of horizontal segments to tracks



- Channel routing (see section 10, page 83)



9. A note on path construction/path algebra

This is a supplement to the shortest path construction. This section can be skipped and the reader can proceed directly to section 10 on channel routing.

Path algebra on a graph $G = (V, E, d_{ij})$ can be described by matrix operations. In the following, we will sketch the three basic operations

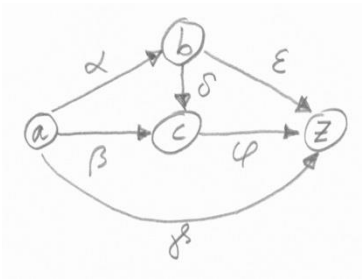
- Path existence,
- Shortest paths,
- Path enumeration,

using the following example:

$$V = \{a, b, c, z\}$$

$$E = \{(a,b), (a,c), (a,z), (b,c), (b,z), (c,z)\}$$

$$d_{ab} = \alpha, d_{ac} = \beta, d_{az} = \gamma, d_{bc} = \delta, d_{bz} = \varepsilon, d_{cz} = \varphi$$



The graph G can als be described by an adjacence matrix $\boldsymbol{P}$. $\boldsymbol{P}$ of the example is:

$$\boldsymbol{P} = \begin{matrix} a \\ b \\ c \\ z \end{matrix} \begin{bmatrix} 0 & \alpha & \beta & \gamma \\ 0 & 0 & \delta & \epsilon \\ 0 & 0 & 0 & \varphi \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

We can determine the powers of $\boldsymbol{P}$:

$$\boldsymbol{P}^k, k = 1, \dots, |V| - 1$$

In the example we have:

$$\boldsymbol{P}^2 = \begin{matrix} a \\ b \\ c \\ z \end{matrix} \begin{bmatrix} 0 & 0 & \alpha \odot \delta & \alpha \odot \varepsilon \oplus \beta \odot \varphi \\ 0 & 0 & 0 & \delta \odot \varphi \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad ; \quad \boldsymbol{P}^3 = \begin{matrix} a \\ b \\ c \\ z \end{matrix} \begin{bmatrix} 0 & 0 & 0 & \alpha \odot \delta \odot \varphi \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad ; \quad (\boldsymbol{P}^4 = \boldsymbol{0})$$

9.1 Path existence

Path existence can be described by the following algebraic definitions in the multiplication of the graph adjacence matrix:

d_{ij} → Boolean variable: $\begin{cases} \text{true} & \text{if } d_{ij} > 0 \\ \text{false (f)} & \text{if } d_{ij} = 0 \end{cases}$

$\odot$ → logical operation $\wedge$
$\oplus$ → logical operation $\vee$

Then, the ij-the element of matrix $\boldsymbol{P}^k$ denotes if there is a path of order k from node i to node j:

Path of order k: path with k edges

$[P^k]_{ij} = \text{true} \quad \Leftrightarrow \quad$ A path of order k from node i to node j exists

In the example we obtain:

$$\boldsymbol{P} = \begin{matrix} a \\ b \\ c \\ z \end{matrix}\begin{bmatrix} \text{f} & \text{true} & \text{true} & \text{true} \\ \text{f} & \text{f} & \text{true} & \text{true} \\ \text{f} & \text{f} & \text{f} & \text{true} \\ \text{f} & \text{f} & \text{f} & \text{f} \end{bmatrix} ; \quad \boldsymbol{P}^2 = \begin{matrix} a \\ b \\ c \\ z \end{matrix}\begin{bmatrix} \text{f} & \text{f} & \text{true} & \text{true} \\ \text{f} & \text{f} & \text{f} & \text{true} \\ \text{f} & \text{f} & \text{f} & \text{f} \\ \text{f} & \text{f} & \text{f} & \text{f} \end{bmatrix} ; \quad \boldsymbol{P}^3 = \begin{matrix} a \\ b \\ c \\ z \end{matrix}\begin{bmatrix} \text{f} & \text{f} & \text{f} & \text{true} \\ \text{f} & \text{f} & \text{f} & \text{f} \\ \text{f} & \text{f} & \text{f} & \text{f} \\ \text{f} & \text{f} & \text{f} & \text{f} \end{bmatrix}$$

9.2 Shortest paths

Shortest paths can be described by the following algebraic definitions in the multiplication of the graph adjacence matrix:

d_{ij} → Path length: $\begin{cases} d_{ij} & \text{if } d_{ij} > 0 \\ \infty & \text{if } d_{ij} = 0 \end{cases}$

$\odot$ → arithmetic addition
$\oplus$ → minimum operation

Then the following information is described by matrix $\boldsymbol{P}^k$:

$[P^k]_{ij}$ = minimum length of path of order k

$\min_k [P^k]_{ij}$ = shortest path length from node i to node j

In the example we obtain:

$$\boldsymbol{P} = \begin{matrix} a \\ b \\ c \\ z \end{matrix}\begin{bmatrix} \infty & \alpha & \beta & \gamma \\ \infty & \infty & \delta & \varepsilon \\ \infty & \infty & \infty & \varphi \\ \infty & \infty & \infty & \infty \end{bmatrix}; \boldsymbol{P}^2 = \begin{matrix} a \\ b \\ c \\ z \end{matrix}\begin{bmatrix} \infty & \infty & \alpha+\delta & \min[(\alpha+\varepsilon),(\beta+\varphi)] \\ \infty & \infty & \infty & \delta+\varphi \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}; \boldsymbol{P}^3 = \begin{matrix} a \\ b \\ c \\ z \end{matrix}\begin{bmatrix} \infty & \infty & \infty & \alpha+\delta+\varphi \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

9.3 Path enumeration

Path enumeration can be described by the following algebraic definitions in the multiplication of the graph adjacence matrix:

d_{ij} → Edge description $\{(i,j)\}$ or $\{\}$

$\odot \rightarrow \circ$ (concatenation of paths)
$\oplus \rightarrow \cup$ (set union operation)

with: $\circ$ dominates $\cup$, $\{(i,j)\} \circ \{(j,l)\} = \{(i,j)\},\{(j,l)\}$, $\{\ \ \} \circ \{(j,l)\} = \{\}$

Then, the matrices $\boldsymbol{P}^k$ identify all paths in a graph:

$[P^k]_{ij}$	=	edges in path of order k from node i to node j
$\cup_k [P^k]_{ij}$	=	respecctive edges of all paths of order k from node i to node j

In the example we obtain:

$$\boldsymbol{P} = \begin{matrix} a \\ b \\ c \\ z \end{matrix} \begin{bmatrix} \{\} & \{(a,b)\} & \{(a,c)\} & \{(a,z)\} \\ \{\} & \{\} & \{(b,c)\} & \{(b,z)\} \\ \{\} & \{\} & \{\} & \{(c,z)\} \\ \{\} & \{\} & \{\} & \{\} \end{bmatrix}$$

$$\boldsymbol{P}^2 = \begin{matrix} a \\ b \\ c \\ z \end{matrix} \begin{bmatrix} \{\} & \{\} & \{(a,b)\circ(b,c)\} & \{(a,b)\circ(b,z),(a,c)\circ(c,z)\} \\ \{\} & \{\} & \{\} & \{(b,c)\circ(c,z)\} \\ \{\} & \{\} & \{\} & \{\} \\ \{\} & \{\} & \{\} & \{\} \end{bmatrix}$$

$$\boldsymbol{P}^3 = \begin{matrix} a \\ b \\ c \\ z \end{matrix} \begin{bmatrix} \{\} & \{\} & \{\} & \{(a,b)\circ(b,c)\circ(c,z)\} \\ \{\} & \{\} & \{\} & \{\} \\ \{\} & \{\} & \{\} & \{\} \\ \{\} & \{\} & \{\} & \{\} \end{bmatrix}$$

10. Channel routing

The basic situation of channel routing with two layers is illustrated in Figure 52. The channel has an upper and a lower border. These borders are discretized by a grid of pins that are to be connected by signal nets.

The pins are numbered, they form so-called columns. We define that the column number refers to the pin number at the upper channel border; the column number with a prime refers to the pin number at the lower channel border.

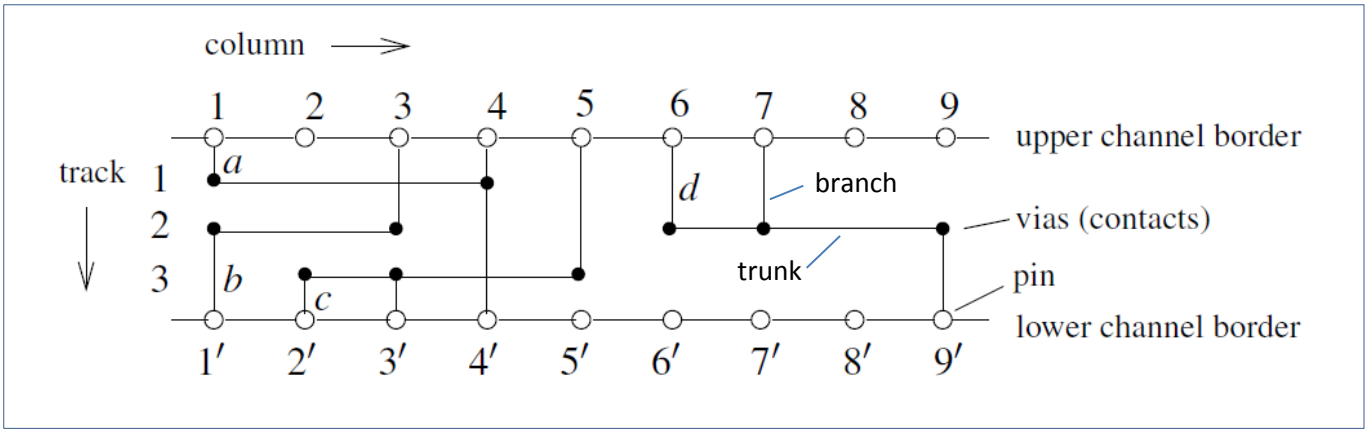
The routing topology of each net consists of one horizontal segment (trunk), which is routed in e.g. layer metal one, and several vertical segments (branches), which are routed e.g. in layer metal two. The horizontal segment of a net is connected to its vertical segments by vias. The wiring of each net has at least one branch at the left end of the trunk and at least one branch at the right end of the trunk.

The channel routing problem refers to a specific Steiner tree construction problem. The general goal is to implement the routing with a minimum number of tracks.

Channel routing is applied in both standard cell and macro cell design (Figure 13, page 15). In macro cell design, the module arrangement forms a basis to partition the space between the modules into a set of channels. A dependency graph is set up that determines the sequence of channels to be routed. The left/right end of one channel forms “pins” for the routing of the next channel. Cycles in the dependency graph are broken by introducing switch boxes.

Looking at Figure 52, we can identify that the **main task of channel routing** consists in **assigning the trunks of the nets to tracks, while satisfying certain constraints**. These are horizontal and vertical constraints, described in the following, using the example shown next.

Figure 52: Channel with two layer routing



10.1 Example

The net-pin list in Figure 53 is an example of a two layer channel routing task. For each net it gives the pin numbers to be connected by this net, where the number denotes the column of the pin. A prime indicates that the pin is located at the lower border of the channel, without prime, the pin is at the upper channel border. The topological routing instruction in Figure 54 can be derived from the net-pin list: The blue lines denote the trunks, the green arrows denote the branches and if they connect to the upper or lower channel border.

Figure 53: Example of a two layer channel routing task: net-pin list

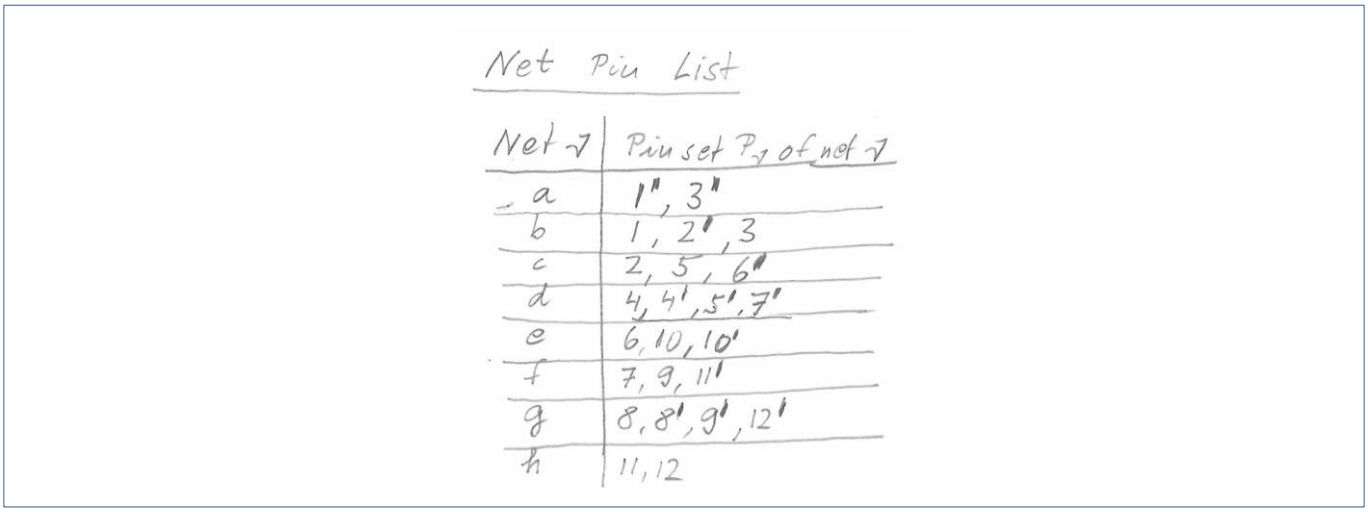
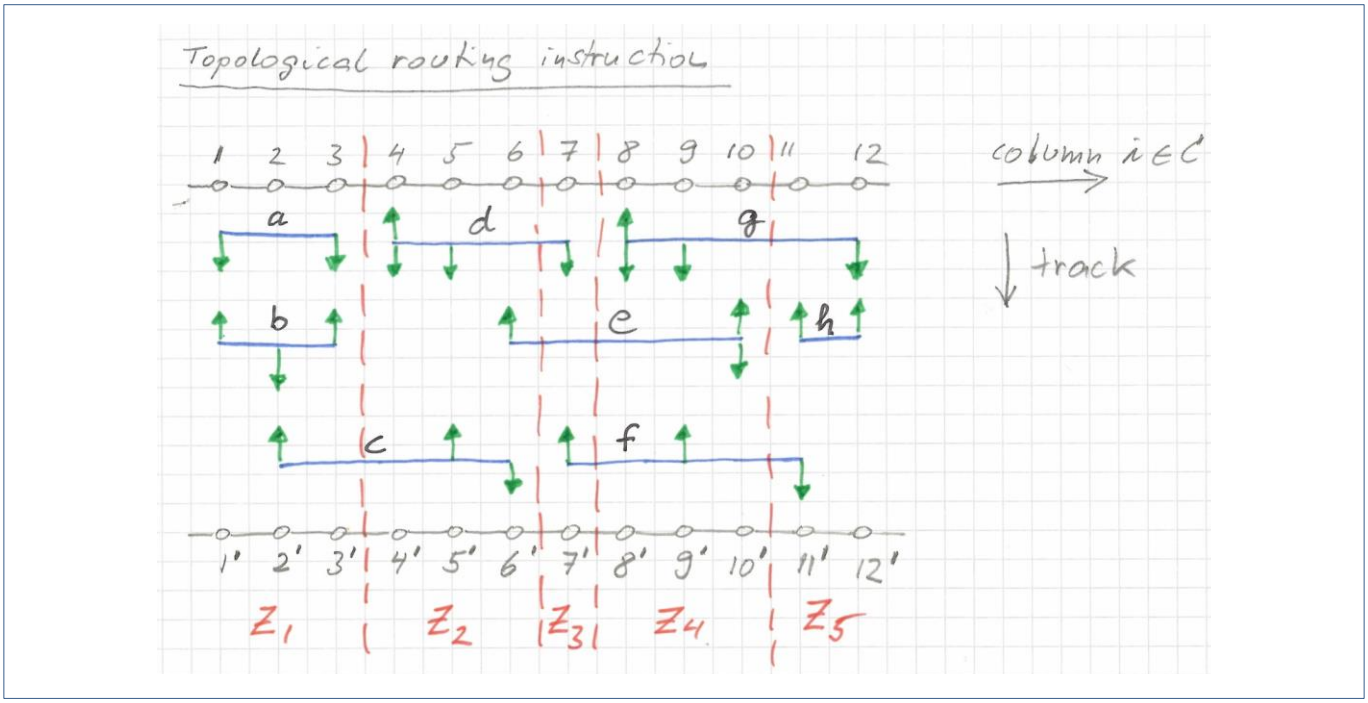


Figure 54: Example of a two layer channel routing task: topological routing instruction



10.2 Horizontal routing constraints

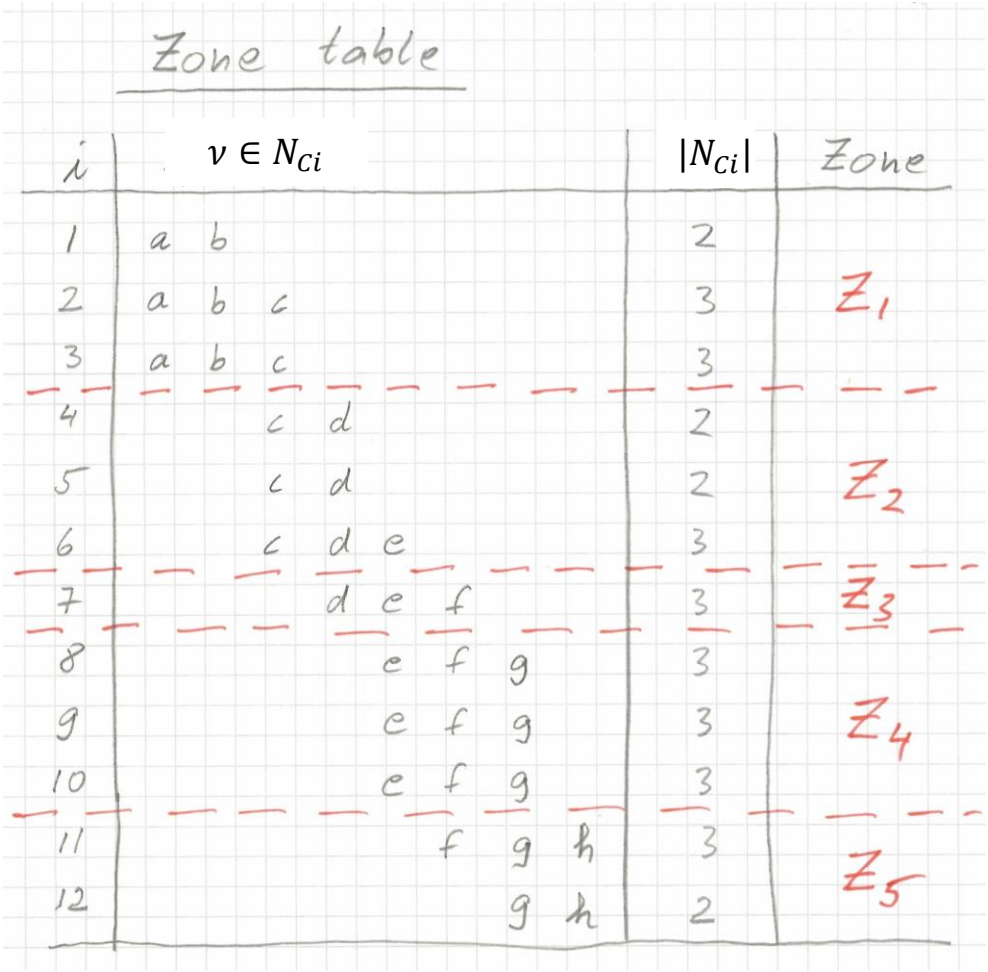
Two nets that cross the same column obviously cannot share the same track, otherwise they would be short-circuited. Horizontal routing constraints can be obtained by going through all columns and setting up a constraint for each pair of nets in a column. Towards this we define the net set P_{Ci} of column i:

Eq. 49 **Net set of column** i : $$N_{Ci} = \{ v \mid \min\{P_v\} \leq i, i' \leq \max\{P_v\} \}$$

Here we have made use of the ordered notation of pins along columns. This means that the trunk of a net spans from the column that corresponds to the minimum pin number of the net's pin set to the maximum number of the net's pin set. For instance, net a goes from column 1 to 3, net b goes from column 1 to 3, net c goes from column 2 to 6, and so on.

The spanning of the trunks over columns can be described as a part of the so-called zone table, as given in Figure 55.

Figure 55: Example of a two layer channel routing task: zone table



Horizontal routing constraints are described by the **horizontal routing constraint graph** $G_H = (N, E_H)$.

An edge $(\nu, \sigma) \in E_H$ in G_H denotes the constraint that the trunks of the two nets ν, σ must be on separate tracks. This is the case if both these two nets appear in the net set of one single column:

Eq. 50 **Single horizontal routing constraint**: $(\nu, \sigma) \in E_H \Leftrightarrow \exists_i (\nu \in N_{Ci} \wedge \sigma \in N_{Ci}) \Leftrightarrow (\sigma, \nu) \in E_H$

We can now go through the net sets of all columns to create edges between all pairs of nets in each column net set. There is a way to structure this process. Let us take a look at Figure 55. Column 1 leads to a horizontal routing constraint between nets a, b, i.e. (a, b). Column 2 leads to constraints between nets a, b, c, i.e. $(a, b), (a, c), (b, c)$. We can see that this set (column, respectively) covers column 1. Therefore we can first search for such columns that cover other columns. These are called dominant columns. Dominant columns refer to certain zones Z that they cover in the sense that the other columns' net sets in that zone are a subset of the dominant column's net set.

Eq. 51 **Dominant column** $i^{(D)}$: $i \text{ is } i^{(D)} \Leftrightarrow \forall_{j \in Z_k} (N_{Cj} \subseteq N_{Ci})$
with Z_k being the zone covered by $i^{(D)}$

Eq. 52 **Zone** Z_k **covered by dominant column** $i^{(D)}$: $Z_k = \{ j \mid N_{Cj} \subseteq N_{Ci^{(D)}} \}$

Please note that according to the problem property of having one trunk per net, a zone is a set of neighboring columns. Please note also that compared to placement, where we met the problem of determining the maximum cliques in a graph, we now determine the maximum cliques (correspond to dominant column net sets) first and then combine them to the overall graph.

For our example, we obtain the following dominant columns and zones, which are shown in Figure 54 and Figure 55:

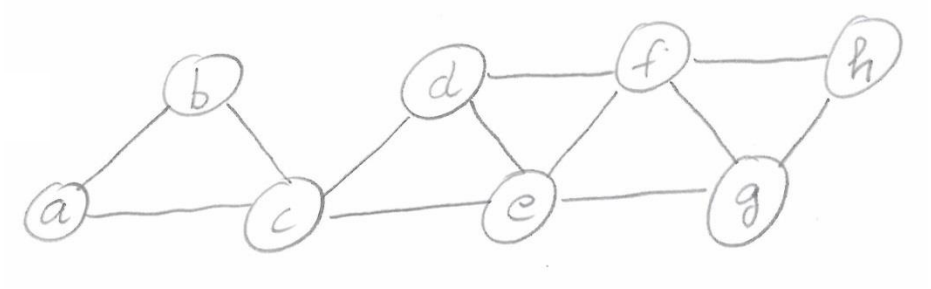
$$Z_1: N_{C2^{(D)}} = \{a, b, c\}$$
$$Z_2: N_{C6^{(D)}} = \{c, d, e\}$$
$$Z_3: N_{C7^{(D)}} = \{d, e, f\}$$
$$Z_5: N_{C8^{(D)}} = \{e, f, g\}$$
$$Z_5: N_{C11^{(D)}} = \{f, g, h\}$$

Each set refers to a clique, or a complete subgraph respectively- These subgraphs are combined to the horizontal routing constraint graph G_H:

Eq. 53 **Horizontal routing constraints:** $E_H = \left[\bigcup_{i^{(D)}} (N_{Ci^{(D)}} \times N_{Ci^{(D)}}) \right] \setminus I$

Figure 56: Example of a two layer channel routing task: horizontal routing constraint graph

G_H:



The horizontal routing constraints result in a minimum number of required tracks for the channel routing. It refers to the number of nets in the largest dominant column set:

Eq. 54 **Minimum number of tracks from horizontal routing constraints:** $U_H = \max_i \mid N_{Ci^{(D)}} \mid$

In the example, we have $U_H = 3$.

10.3 Vertical routing constraints

While horizontal routing constraints result from the trunk part of the wiring of a net, vertical routing constraints result from the branch part of the wiring.

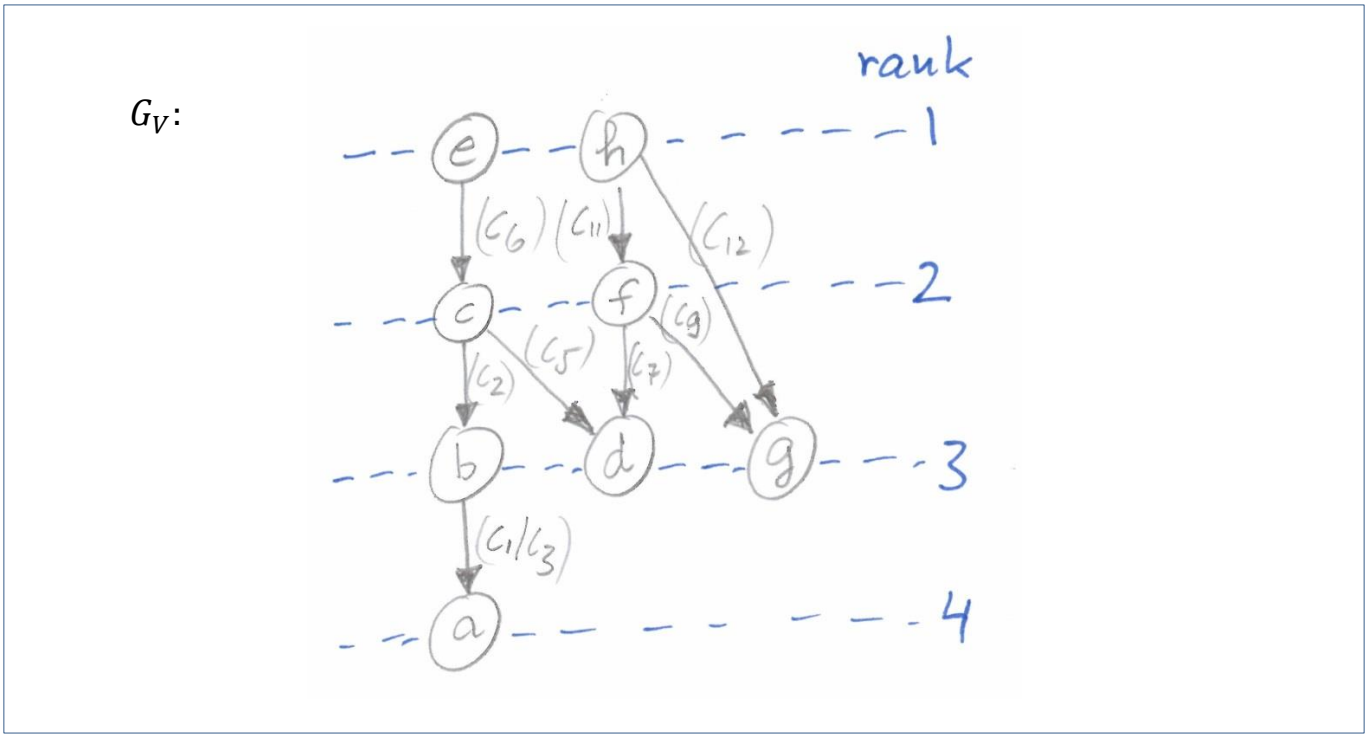
If a net ν is connected to the upper channel border in a column i, and a net σ is connected to the lower channel border in the same column i, then net ν must be assigned to a track **above** the track of net σ, to avoid a short-circuit:

Eq. 55 **Single vertical routing constraint**: $(\nu, \sigma) \in E_V \Leftrightarrow \exists_i (i \in P_\nu \wedge i' \in P_\sigma)$

The resulting **vertical routing constraint graph** $G_V = (N, E_V)$ is a directed graph. A necessary condition for routability is that G_V is **cycle-free**.

The vertical constraints can be obtained from the signal-pin list by searching for i, i'-pairs and adding an edge for the respective nets in G_V. For the example, we obtain the vertical routing constraint graph in Figure 57.

Figure 57: Example of a two layer channel routing task: vertical routing constraint graph



Each vertex in G_V has a rank. Vertices of rank 1 have no predecessors. Vertices of rank 2 have no predecessors if rank-1 vertices and their incident edges have been removed, and so on. Another definition is:

Rank of a vertex in G_V: Number of nodes in the longest path to this vertex.

The ranks of vertices allow to determine the minimum number of required tracks for the channel routing due to vertical routing constraints. It refers to the maximum rank in G_V:

Eq. 56 **Minimum number of tracks from vertical routing constraints:** $U_V =$ maximum rank in G_V

In the example, we have $U_V = 4$. It refers to the result that due to the branches, net e has to be on top of c, which has to be on top of b, which in turn has to be on top of a.

Please note that the vertical routing constraint graph is a subset of the horizontal routing constraint graph. This can be seen from the following:

If a net ν connects to the upper channel border in column i, and a net σ connects the the lower channel border in column i, then it holds that both nets ν, σ are in the net set of column i, i.e.,

$$i \in P_\nu \ \wedge i' \in P_\sigma \Longrightarrow \ \nu \in N_{Ci} \ \wedge \sigma \in N_{Ci}$$

This is equivalent to saying that if (ν, σ) is a vertical routing constraint, then it is also a horizontal routing constraint, i.e,

$$\Leftarrow\Rightarrow \qquad (\nu, \sigma) \in \ E_V \ \Longrightarrow \ (\nu, \sigma) \in \ E_H$$

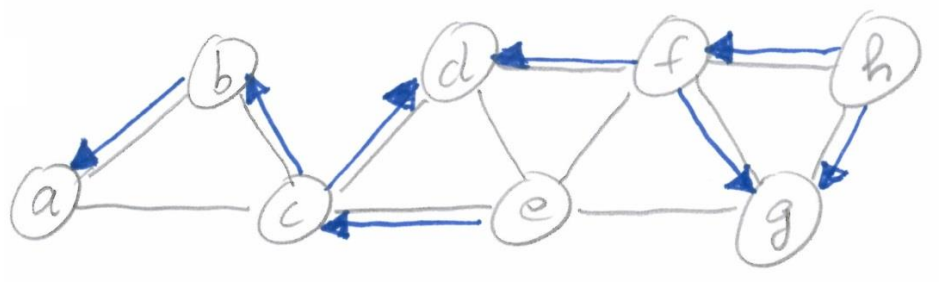
This in turn describes that the vertical routing constraint graph is a subset of the horizontal routing constraint graph, i.e,

$$\Leftarrow\Rightarrow \quad E_V \ \subseteq E_H$$

Therefore, we can "draw" G_V into G_H to obtain the pseudograph of routing constraints. For the example, we obtain the pseudograph in Figure 58.

Figure 58: Example of a two layer channel routing task: pseudograph of routing constraints

G_P:



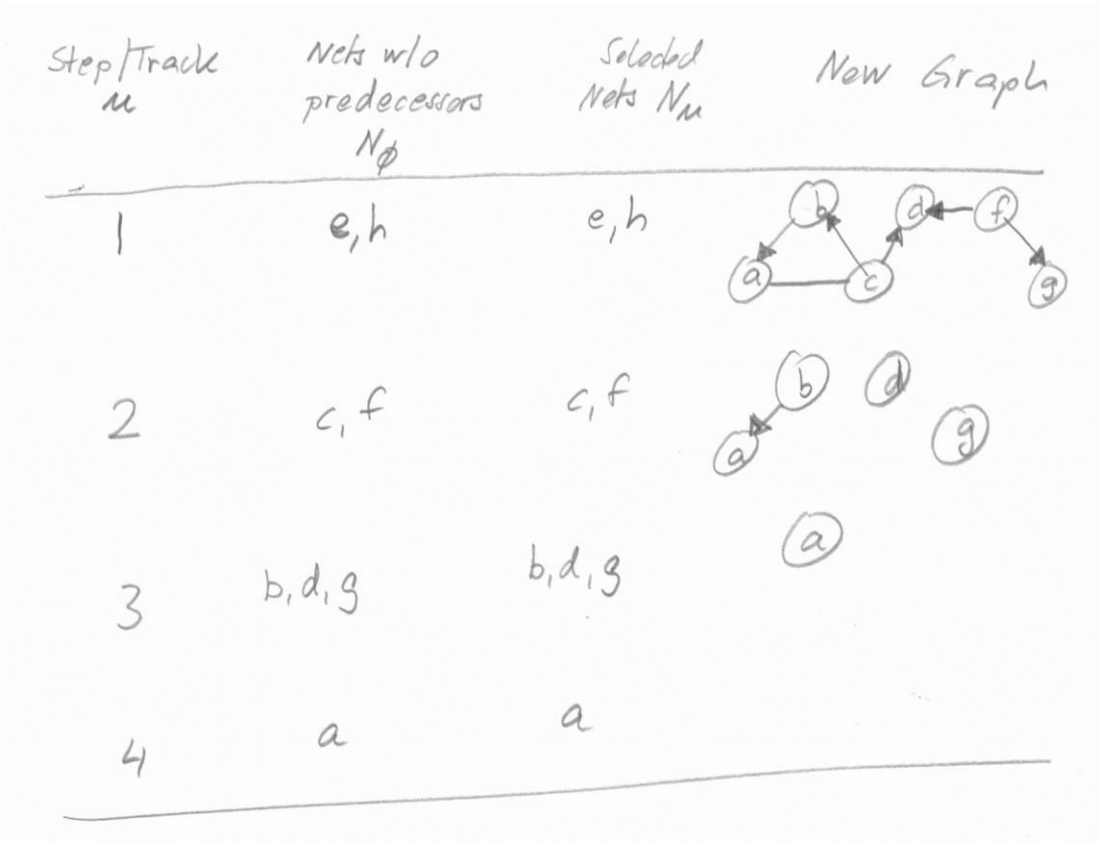
10.4 Heuristic Left-Edge algorithm for channel routing

The basic task of channel routing is the assignment of trunks to tracks. The constraints are collected in the horizontal and vertical constraint graph, as done in the pseudograph in Figure 58.

The objective is to minimize the number of required tracks. To achieve this goal, we start from the vertices in the pseudograph without predecessors and assign them to the first (topmost) track, provided that they do not have any horizontal constraints. Then we delete these vertices, increment the track under consideration and repeat the process until all vertices are done, i.e., all nets are assigned to tracks. This is the basic principle of the Left-Edge algorithm. In case of horizontal constraints among the vertices without predecessor, heuristics have to be applied to select a subset without horizontal constraints.

For the example we get the following process, starting from the pseudograph in Figure 58. In the first step, nets e and h are selected as they have no predecessors. They have no horizontal constraints, therefore are assigned to track 1. Then, we delete the corresponding vertices and their adjacent edges and obtain the graph as given in Figure 59.

Figure 59: Left Edge process on example



The algorithm is depicted in Figure 60. (b) and (c) are specific heuristics for channel routing. (b) takes care that nets carrying constraints that lead to a large number of tracks are assigned as soon as possible. (c) takes care to fill tracks as much as possible.

The resulting channel routing is given in Figure 61.

Figure 60: Left-Edge algorithm "flavored" with heuristic for channel routing

Initially, $u := 0$ (track number

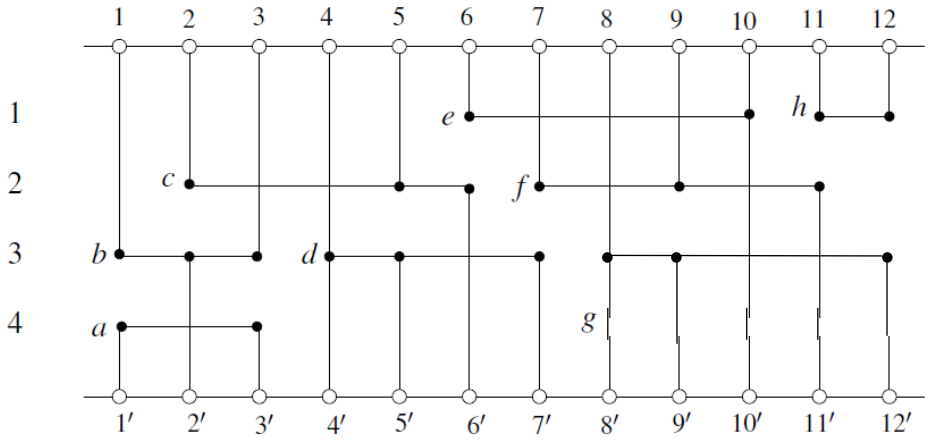
Repeat

$u := u + 1$
$N_0 :=$ set of vertices (nets) without predecessors in G_V
If no horizontal restrictions among nets in N_0
$N_u = N_0$ (set of nets on track u)
Else
Determine $N_u \subset N_0$ such that
(a) No horizontal constraint violated
(b) Nets are on long paths in G_V and in large cliques of G_H
(c) $|N_u|$ is large
Endif

"Problem reduction":
Delete vertices in N_u and adjacent edges from pseudograph

Until all nets assigned to tracks

Figure 61: Final channel routing of example



10.5 Channel routing with doglegs in interior pins of nets

If we take a closer look at Figure 61, we can see that track 4 is required only for the three columns used by the trunk of net a. And we see that track one is “free” from column 1 to column 5. If we could push net c from track 2 to 1 in columns 1 to 3, and push net b from track 3 to 2, we would be able to move net a to track 3 and delete track 4. Figure 62 shows how to do this.

The revised approach of channel routing is that the **trunk of a net may be split and distributed on more than one track**. Due to resulting shape of the trunk this is called “dogleg”.

Another motivation for doglegs is a cycle that may appear in the vertical constraint graph. Figure 63 illustrates this situation. We can see that net a should be on top of net b according to column 1, and net b on top of a according to column 3. By allowing a dogleg for net a in column 2, the cycle disappears and the channel can be routed using three tracks.

While doglegs may help saving tracks in regular situation, they may help solve cycles at the cost of extra tracks.

Figure 62: Channel routing of example with doglegs

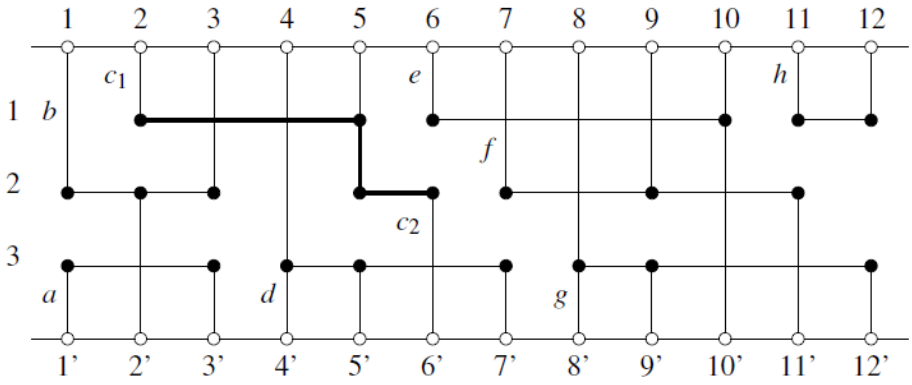
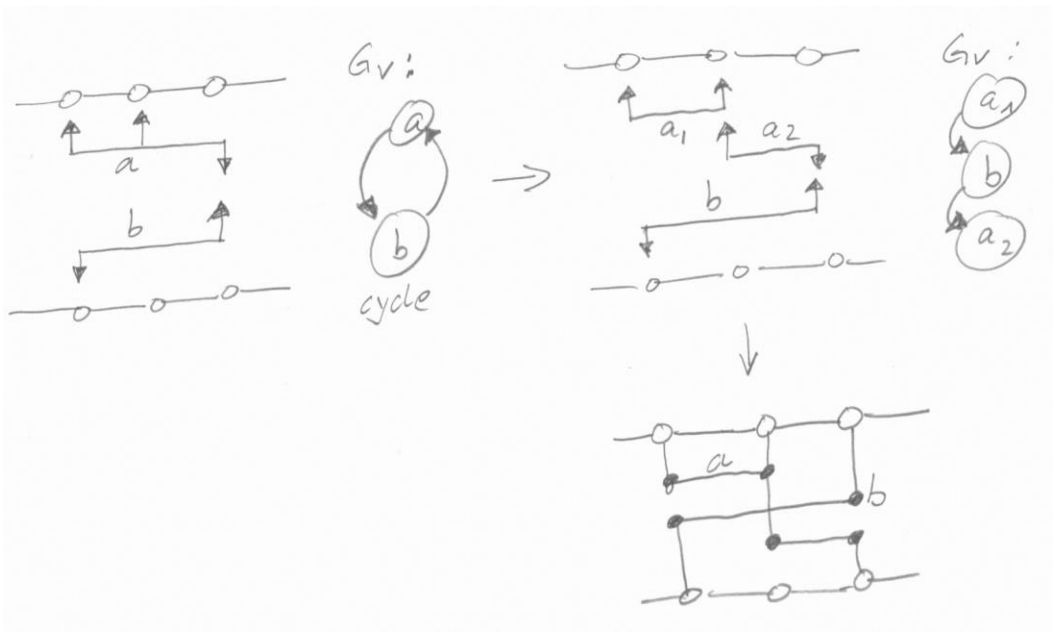


Figure 63: Cycle in vertical constraint graph and solution by dogleg



Purpose of doglegs

I. Reduction of overall number of tracks
II. Resolution of cycles in vertical constraint graph

Potential location of doglegs

We allow doglegs potentially at interior pins of a net v:

Eq. 57 **Interior pin set of net** v: $$P_{\text{int},v} = \{\, i \mid i, i' \in P_v \ \wedge \ \min\{P_v\} < i, i' < \max\{P_v\} \,\}$$

An interior pin of a net is a pin to the upper or lower border of the channel that is *not* the leftmost or rightmost end of the trunk of a net. This means that we potentially allow that the trunk of a net is allowed to be split onto two tracks at some existing pin in between the left and right end.

Please note that this is the basic approach to doglegs. It can be extended to allowing a split at a column where the trunk has no predefined pin, and it can be extended splitting on more than 2 tracks.

Conditions for split

I. If a reduction of required number of tracks is targeted, then there should be at least two free tracks in the column where a net is split: one track is for splitting the net, which then occupies two instead of one track in that column, the other track will be eventually deleted as part of the track saving process:

Eq. 58
$$|N_{Ci}| \leq \begin{cases} U_H - 2 & \text{after horizontal constraint graph} \\ \max\{U_H, U_V\} - 2 & \text{after horizontal and vertical constraint graph} \\ u - 2 & \text{after track assignment} \end{cases}$$

Note that the net split can be done after

- set up of the horizontal constraint graph,
- set up of horizontal and vertical constraint graph,
- track assignment.

This is reflected in the three cases in Eq. 58.

II. If a net is in a cycle of the vertical constraint graph, it will be split.

In the example of Figure 53, we will check condition in Eq. 58 after having set up the horizontal and vertical constraint graphs. From Figure 54 and Figure 55, we can see that columns 2, 3, 6-11 only have one free track and that columns 1, 4, 12 do not have interior pins. Only/precisely in **column 5** we have 2 free tracks, as $|N_{Ci}| = 2$ and $\max\{U_H, U_V\} = 4$, and 2 **signals** $\boldsymbol{c}$, $\boldsymbol{d}$ with interior pins.

This leads to a revised split net pin list as given in Figure 64, which corresponds to the revised topological routing instructions as given in Figure 65.

We can see that the introduction of two signal parts with individual names lead to a property that be an error in the original netlist: two nets connect to the same pin. This property has to be considered in the further proceeding of channel routing.

We can now define a property that identifies split nets in the revised netlist. Split nets can be recognized through the revised pin sets: unlike regular nets, the intersection of pin sets of split nets now is not empty (because they connect to some common pin):

Eq. 59
$$\textbf{Nets } \nu, \sigma \textbf{ are split nets} \quad \Leftrightarrow \quad P'_\nu \cap P'_\sigma \neq \{\}$$

If nets are split, then there are no horizontal and no vertical constraints at the column where the split is done. This is framed in red in Figure 64 and Figure 65. In column 5, c_1 and c_2 may, but do not have to be put on separate tracks, we only allow the assignment to different tracks, but do not enforce it. It is left to the algorithm where they are put. Analogously, it is left open on which track they might come to lie, so there is also not vertical constraint at this point. The same holds for d_1 and d_2 in the example.

Figure 64: Example with doglegs: split net-pin list

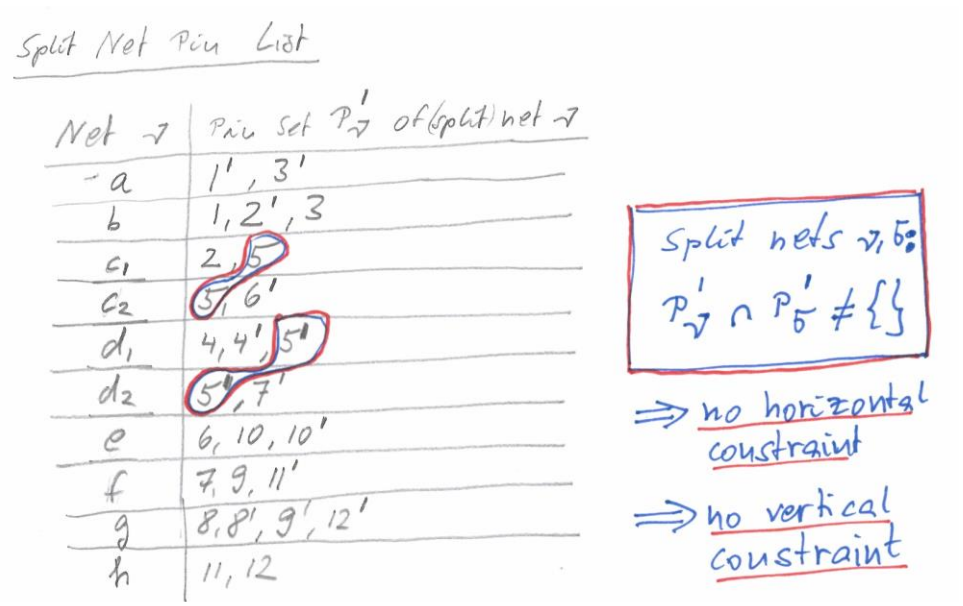
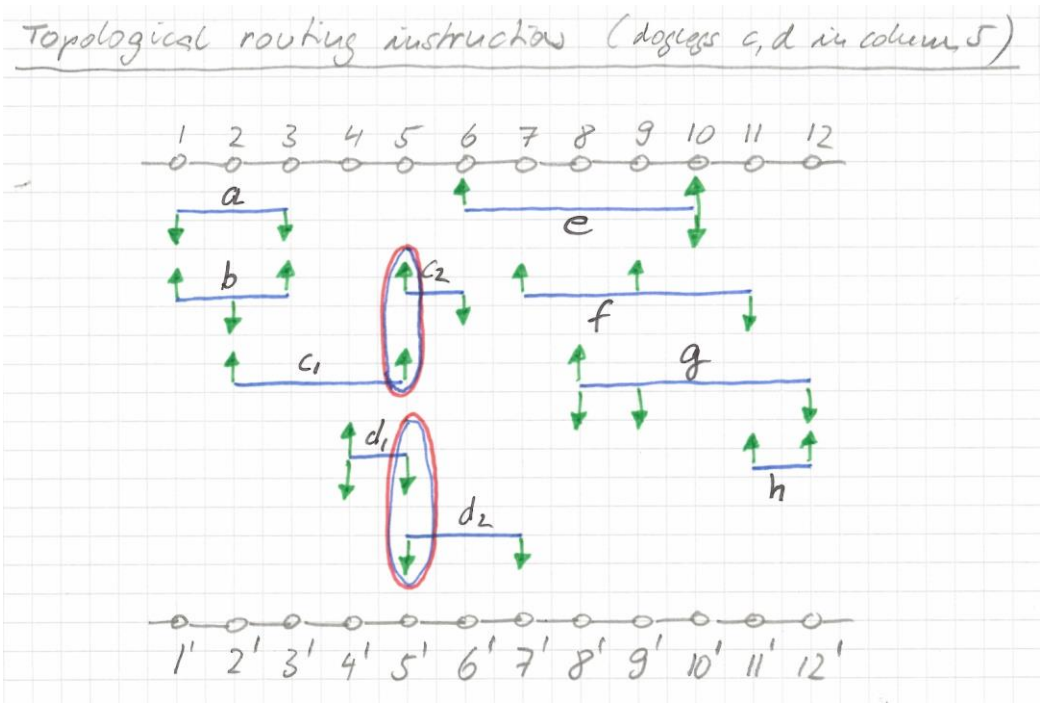


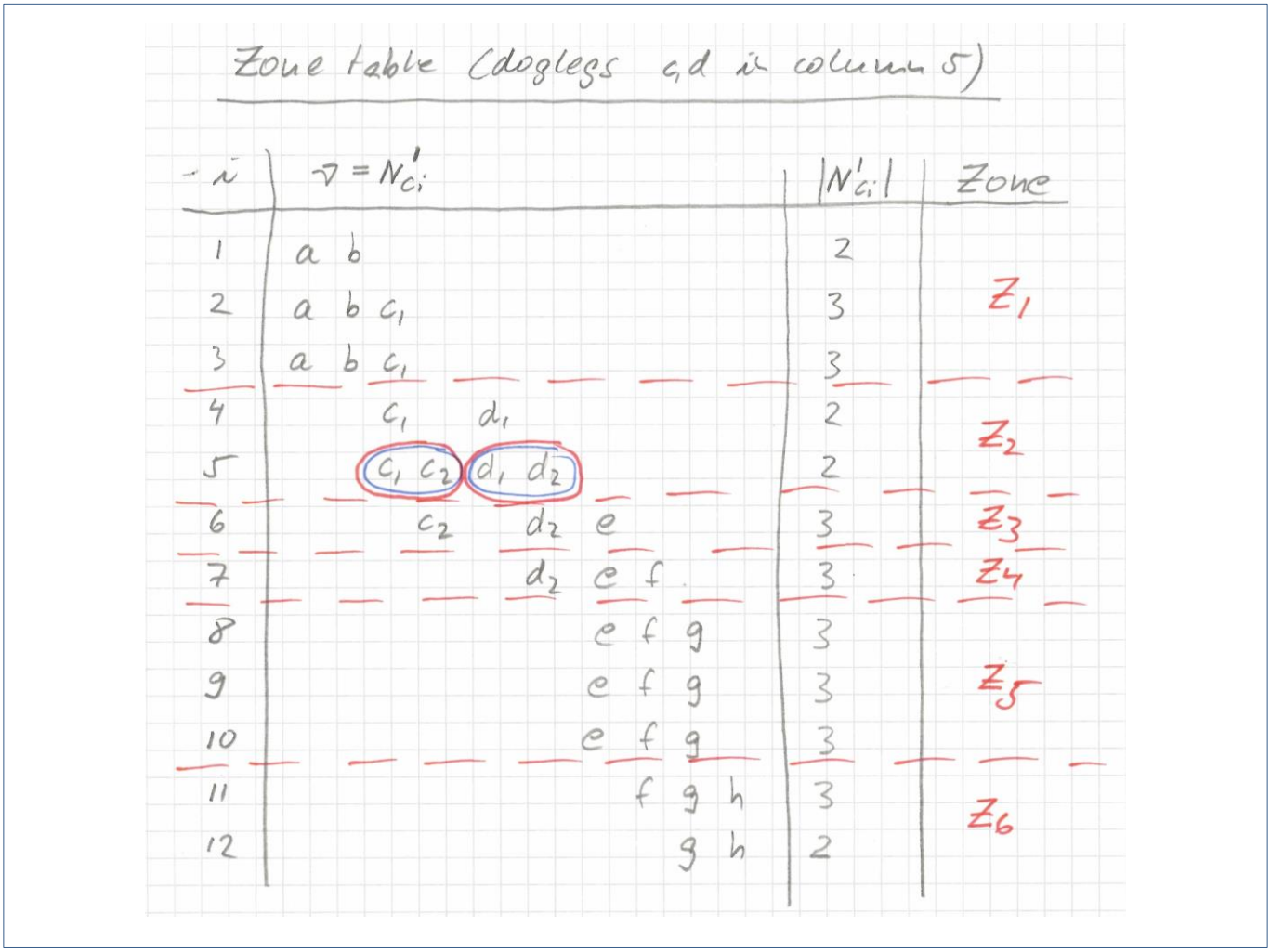
Figure 65: Example with doglegs: topological routing instruction



Set up revised zone table

Next, the revised zone table is computed. For the example, we obtain the result in Figure 66.

Figure 66: Example with doglegs: zone table



Set up revised horizontal and vertical constraint graph

Next, the revised horizontal and vertical constraint graphs are computed. The rules are as described in Eq. 50 and Eq. 55, with the difference exclude the split nets (Eq. 59):

Eq. 60 **Single horizontal routing constraint (with doglegs)**:

$$(v,\sigma) \in E'_H \quad \Leftrightarrow \quad \exists_i \, (v \in N_{Ci} \, \wedge \sigma \in N_{Ci} \, \wedge \, P'_v \, \cap P'_\sigma = \{\}) \quad \Leftrightarrow \quad (\sigma, v) \in E'_H$$

Eq. 61 **Single vertical routing constraint (with doglegs)**:

$$(v,\sigma) \in E'_V \quad \Leftrightarrow \quad \exists_i \, (i \in P_v \, \wedge i' \in P_\sigma \wedge \, P'_v \, \cap P'_\sigma = \{\})$$

For the example, we obtain the result in Figure 67 and Figure 68.

Figure 67: Example with doglegs: horizontal routing constraint graph

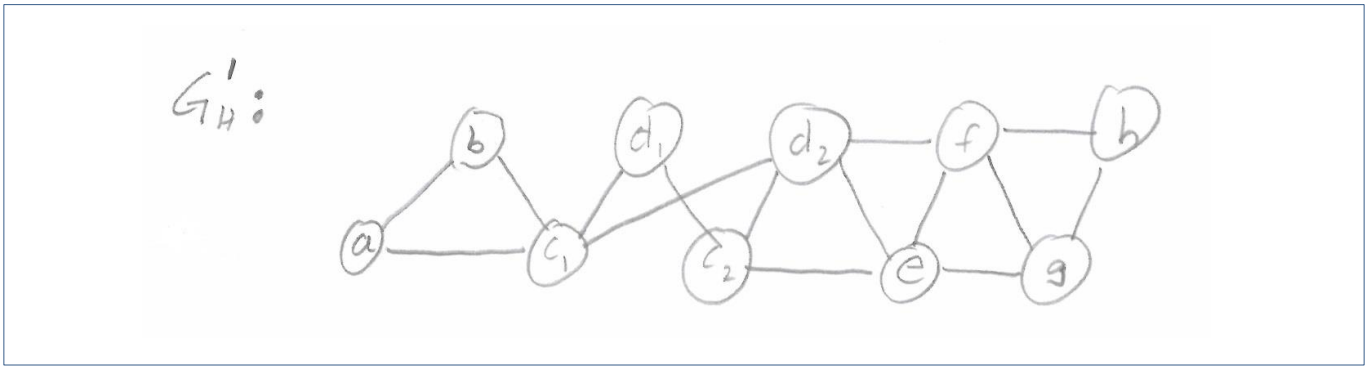
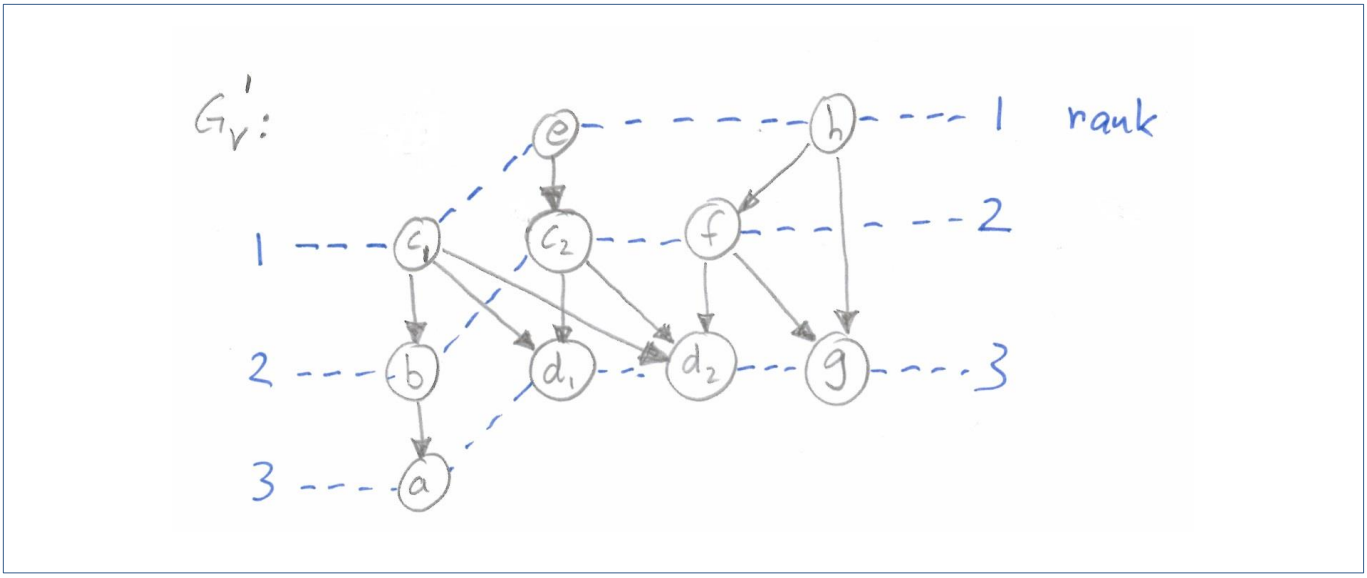
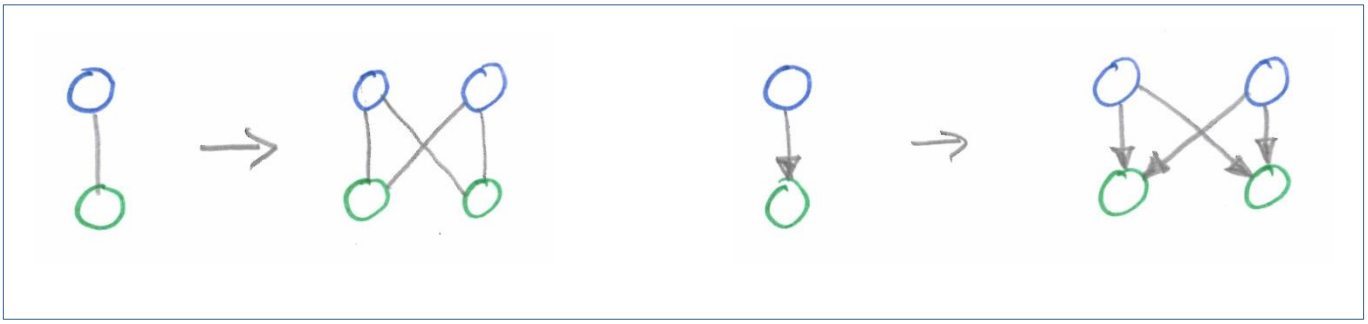


Figure 68: Example with doglegs: vertical routing constraint graph



Comparing the horizontal/vertical constraint graphs with and without doglegs (Figure 67/Figure 68 with Figure 56/Figure 57), we can derive the general rule how splitting nets maps into a revision of the horizontal and vertical constraint graph (Figure 69).

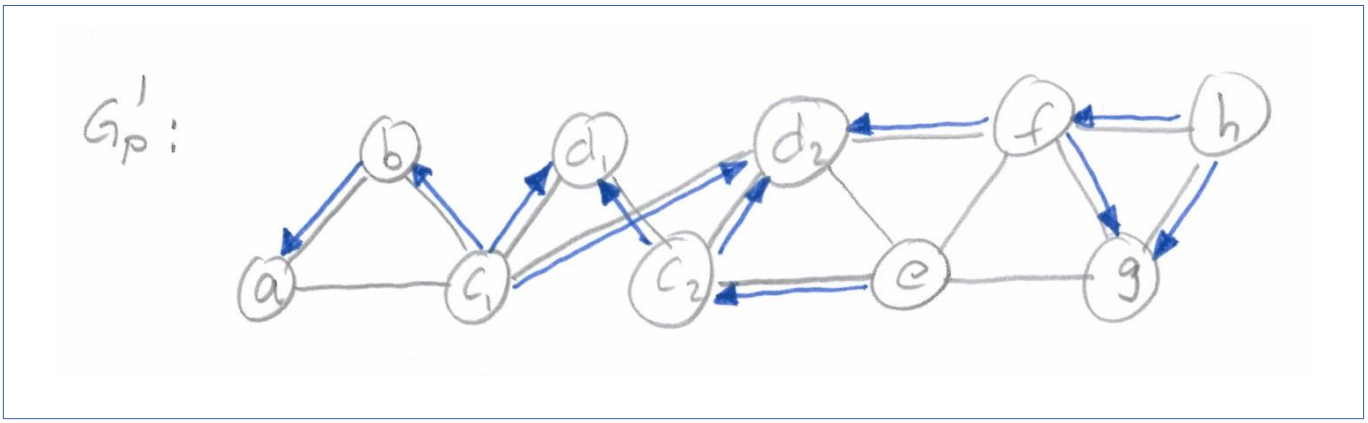
Figure 69: Mapping a net split into horizontal and vertical routing constraint graph



Set up revised pseudograph of routing constraints

The revised constraint graphs are merged to the revised pseudograph of routing constraints. For the example, Figure 67 and Figure 68 yield Figure 70:

Figure 70: Example with doglegs: pseudograph of routing constraints

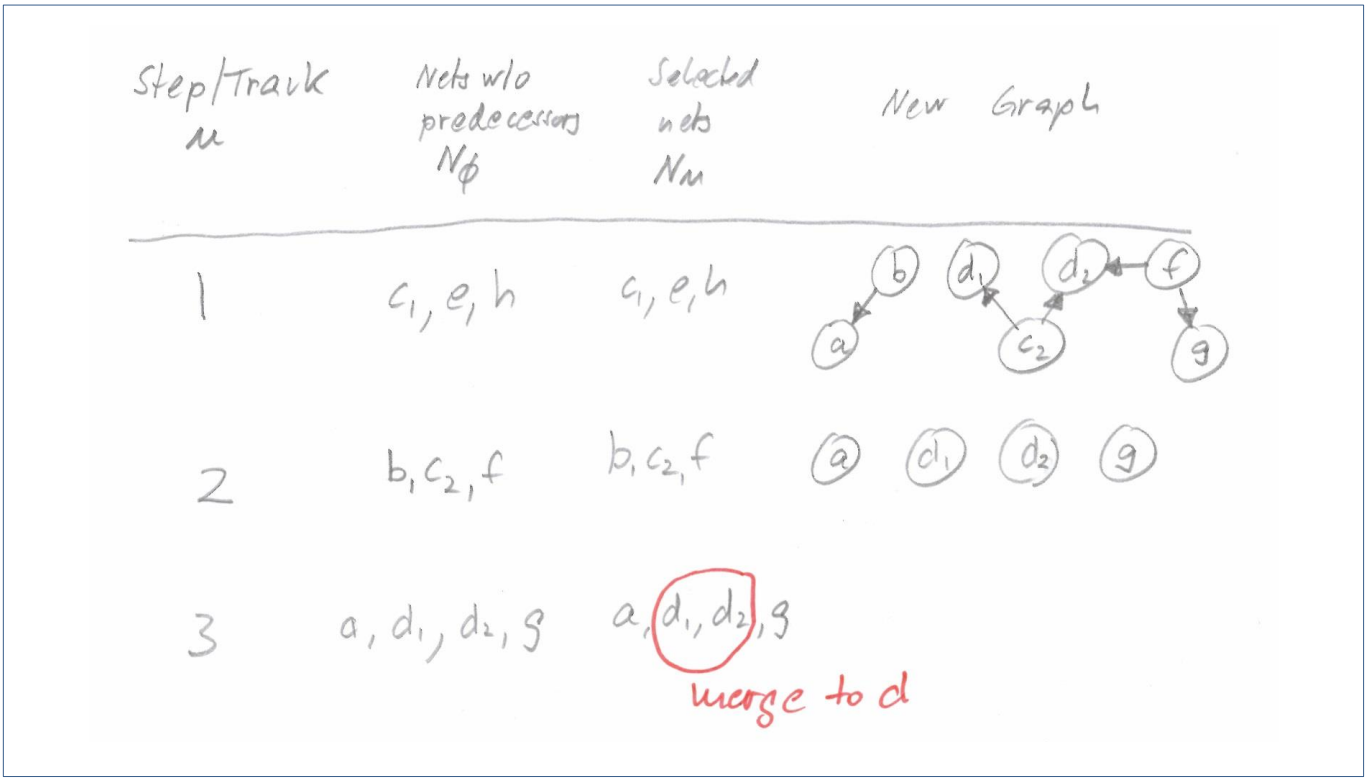


Revised track assignment

Track assignment is then revised using the same algorithm of Figure 60, now for the pseudograph. For the example, the process is as given in Figure 71.

We can see that in fact nets c_1 and c_2 are assigned to different tracks, whereas nets d_1 and d_2 come to lie on the same track and hence are merged back to one net d.

Figure 71: Left Edge process on example with doglegs



11. Literature

Basics:

A. V. Aho, J. E. Hopcroft, J. D. Ullman: *Data Structures and Algorithms*, Addison-Wesley Publishing Company, Reading, 1985.

P. E. Gill, W. Murray, M. H. Wright: *Practical Optimization*, Academic Press, London, 1982.

P.A. Jensen and J.W. Barnes: *Network Flow Programming*, John Wiley & Sons, Inc. 1980.

M. Grötschel, L. Lovász, A. Schrijver: *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin Heidelberg, 1988.

F. P. Preparata, M. I. Shamos: *Computational Geometry*, Springer-Verlag, 1985.

P. J. M. van Laarhoven, E. H. L. Aarts: *Simulated Annealing: Theory and Applications*, Reidel Publishing, Dordrecht, 1987.

T. Cormen, C. Leiserson, and R. Rivest: *Introductions to Algorithms*, MIT Press, 4. printing 1994.

N. Wirth: *Algorithms + Data Structures = Programs*, Prentice-Hall series in automatic computation.

Journals, Conferences:

IEEE Transactions on Computer-Aided Design of Circuits and Systems

ACM/IEEE Design Automation Conference (DAC)

IEEE International Conference on Computer-Aided Design (ICCAD)

ACM/IEEE International Symposium on Physical Design (ISPD)

Textbooks Layout:

E. Hörbst, M. Nett, H. Schwärtzel: *VENUS: Entwurf von VLSI-Schaltungen*, Springer-Verlag, Berlin, 1986.

T.C. Hu, E.S. Kuh (Editors): *VLSI Circuit Layout: Theory and Design*, IEEE Press, New York, 1985.

T. Lengauer: *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1990.

B. Preas, M. Lorenzetti: *Physical Design Automation of VLSI Systems*, Benjamin/Cummings Publishing Company, California, 1988.

J. Lienig: *Layoutsynthese elektronischer Schaltungen (in German)*, Springer, 2006.

Papers Layout:

A. Kuehlmann (Editor): *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*, Kluwer Academic Publishers, 2003.

E.W. Dijkstra: *A Note on Two Problems in Connexion with Graphs*, Num. Mathematics, Vol.1, 1959.

K. Doll, F.M. Johannes and K.J. Antreich: *Iterative Placement Improvement by Network Flow Methods*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD, October 1994.

F. Rubin: *The Lee Path Connection Algorithm*, IEEE Transactions on Electronic Computers, September 1974.

L. Steinberg: *The Backboard Wiring Problem: A Placement Algorithm*, SIAM Review, January 1961.

T. Yoshimura: *An Efficient Channel Router*, ACM/IEEE Design Automation Conference (DAC), 1984.

12. Index