What is a Floorplan?

Ralph H.J.M. Otten
Eindhoven University of Technology, Eindhoven, The Netherlands
otten@ics.ele.tue.nl

ABSTRACT

This paper surveys the more important floorplan models and their use in the backend of chip design. Floorplans are here considered as data structures that capture the relative positions of objects in a plane. This is in correspondance with how they were introduced in design automation in the early eighties: as a generalization of placement, that is the manipulation of geometrically fixed objects in order to get an overlap-free arrangement of these objects in a plane region. The introduction of the floorplanning concept not only liberated back-end design from its (manual) puzzle character and its intractable and sometimes even undecidable problems, but also enabled hierarchical design, even taking a functional hierarchy and transforming it by stepwise refinement into a floorplan with a compatible representation. The elegance of that procedure was so overpowering that many identified floorplanning with hierarchical back-end design. That floorplanning proves to be a powerful notion by itself becomes once more apparent, since almost all modern effective "*placement*" procedures are in essence floorplanning followed by packing or some other "legalization" stage.

1. INTRODUCTION

Over the years floorplanning became strongly associated with hierarchy, or hierarchical design. Yet, its original meaning was simply a generalization of placement. Whereas placement was the manipulation of geometrically fixed objects to arrange them in a configuration without overlaps, floorplanning handled objects of which the shape did not have to be fixed. A *floorplan* was simply a data structure capturing the relative positions of the objects. Floorplan optimization was a procedure to produce a rectangle dissection compatible with the given floorplan and optimal with respect to a given objective while each object had a feasible shape.

Floorplanning lent itself much better for top down approaches where the shape and sometimes even the precise size of the modules were unknown. Certain restraints enabled *stepwise refinement* in that the tree representation of the floorplan was forced to be a refinement of a given hierarchy tree. These features caused the strong association of the floorplan concept with hierarchical design.

Today it can safely be stated that placement has been replaced by floorplanning, followed by some legalization step. Modern back-end tools almost exclusively start with generating point configurations (with or without size-dependent spacing) or overlapping geometrical objects, mostly by analytic optimizers that can handle complex but flat incidence structures with library elements of fixed shape. Once the optimum for this configuration is established, legalizers ensure that all overlap is removed, and that routability is enhanced. Ironically, this was exactly what early fully automatic floorplanners did [13].

In this paper we work from the definition of a floorplan to the many models and configurations that have served to capture relative positions in one way or the other. We emphasize the importance of meaningful and easy to handle metrics for evaluating floorplans, although the in-depth treatment and comparison is left to another paper at this conference.

2. FLOORPLANS

Formally, a floorplan is a data structure that captures the relative positions of objects which get their final shape by optimizing some objective without violating constraints.

This definition does not imply any underlying hierarchical structure. It simply generalizes the notion of placement: the manipulation of objects with fixed geometrical features in order to allocate them without any overlap in an enclosing plane region. The generalization refers to both, the uncertainty or flexibility of shape of some or all of the objects, and the possibility of overlap when these objects have a shape or area associated.

It is evident of course, that hierarchical approaches, that want to find regions for allocation of subsets of modules almost have to resort to floorplanning in one way or the other, because it is neither always wise nor often feasible to predetermine the shape of all modules in a hierarchy. This obvious need for floorplanning in approaches that one to preserve, completely or partly, properties of earlier partitioning, caused an almost identification of floorplanning with hierarchical layout design. The elegant formulation of stepwise refinement for layout synthesis, where preserving the hierarchical structure of a functional design occurred auto-

ISPD 2000, San Diego, CA.

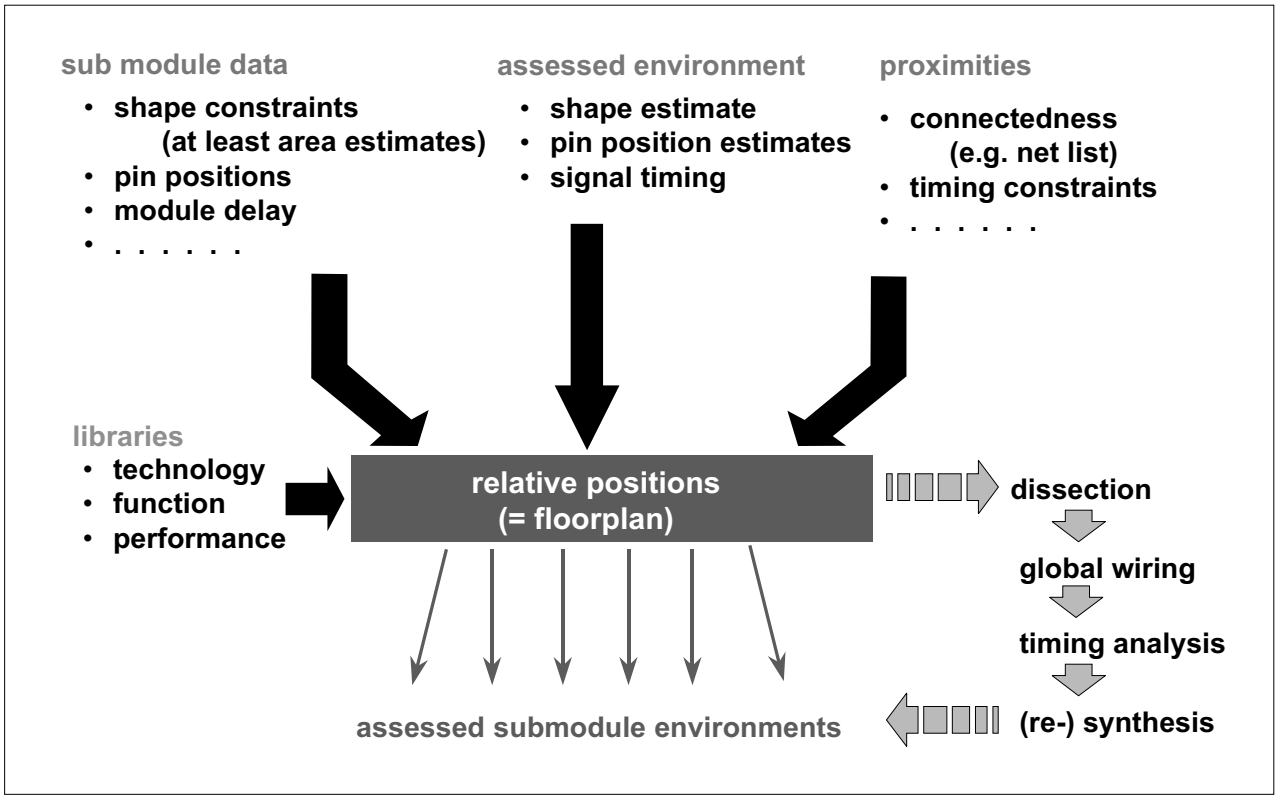

Figure 1: Floorplan design

matically by adopting a structural restraint that expanded, ordered and labeled the original hierarchy tree, planted this idea firmly in the minds of back-end tool developers of the late eighties. Central was the introduction of the concept of a floorplan in the following way:

A floorplan is a data structure that captures the relative positions of modules in a supermodule

- derived from
  - a preliminary shape of the supermodule
  - a preliminary environment (e.g. pin positions)
  - shape information for the modules (e.g. shape constraints)
  - an incidence structure

  and possibly
  - timing information of external signals
  - module delay information
  - path delay information
- used for
  - creating a preliminary environment
  - assessing wire space requirements
  - estimating the delays
- controlled by the (given or derived) hierarchy

Figure 1 illustrates this definition, and more or less invites the nested application of the central floorplan-call. The terminology, the creation and use of module environments, and the explicit mention of hierarchy in the last item, seems to bind floorplanning exclusively to hierarchical design. However, a flat design with a single supermodule and a single level of leaf cells (undivided modules) is perfectly consistent, and was always the most important practical application of the floorplanning concept. I even believe that it has rendered classical placement obsolete, because of its flexibility and relative efficiency.

In this section we look at three classes of floorplans, not necessarily in chronological order of origin, but fairly representative of what has been conceived over three decennia for capturing relative positions of modules.

2.1 Graphs

The oldest floorplans are graphs. Drawing a rectangle dissection already produces a graph, where line segments are interpreted as edges and the points where these come together (mostly *T-junctions*, sometimes *crossings*) as vertices. It should be noted that different dissections may lead to the same graph. A special property that such a graph should have is that its edges can be divided into two classes, *h-edges* and *v-edges*, and it possesses a plane representation where these edges are represented by orthogonal line segements, one orientation per class. This graph is called the *floorplan graph* (figure 2a).

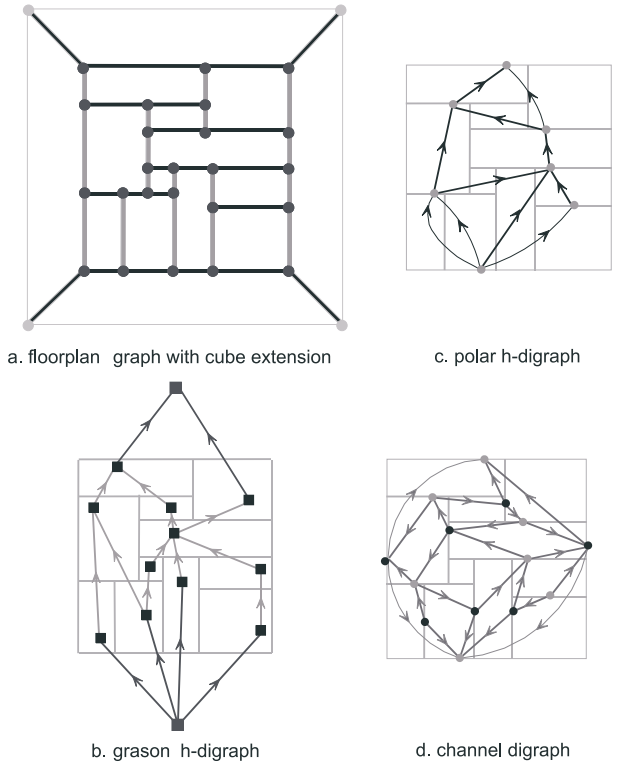

Figure 2: Graphs

The first formal floorplanning procedure started with a graph in which the edges represented desired adjacencies. The vertices represent the objects, that is modules, but in the first applications, *rooms*. Obviously, not every such graph can be converted into a rectangle dissection. They should be planar, but that is not enough. In addition they should have a dual that has a plane representation with two orthogonal sets of line segments, as described above. Graphs with such a dual are called *grason graphs* in honour of the first scientist to publish a floorplanning procedure [5]. This pioneer also gave a characterization of grason graphs, but efficient algorithms for recognizing grason graphs and constructing an associated rectangle dissection came some fifteen years later [9, 1, 2, 17]

Based on the division into h- and v-edges we can also divide the edges of the grason graph, yielding two graphs: the *grason-h-graph* and the *grason-v-graph*. By directing these graphs consistently with the relative positions of the rectangles of their vertices we obtain *grason-h-digraph*(figure 2b) and the *grason-v-digraph*.

Equivalent to this digraph pair (or the colored grason graph), and also to the colored floorplan graph (that is, with explicitly given h- and v-edge-set) is the *channel digraph* (figure 2d) where each channel (i.e. maximal line segment) has a vertex and each T-junction has an arc pointing to the "bar of the T" [3]. This graph has hardly been used though it has some characteristic properties and retains generality where the much more popular *polar digraphs* are not capable of capturing all adjacencies.

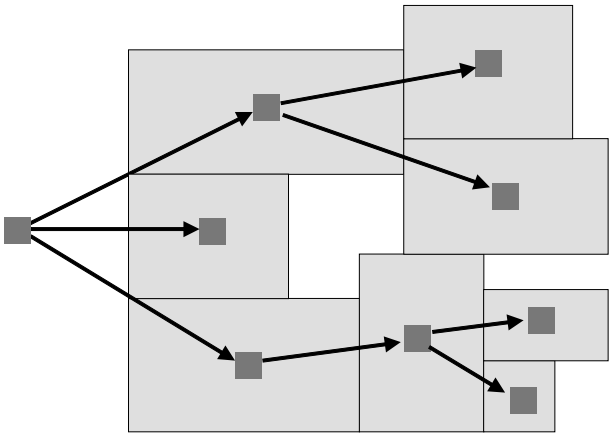

Figure 3: Labeled ordered tree

Although known already in the first half of the twentieth century, these polar graphs were introduced [12] into layout synthesis in 1970 by Tatsuo Ohtsuki, the japanese circuit theorist to whom we owe so many graph models for layout problems. The *polar h-digraph* (figure 2c) takes the maximal horizontal line segments as its vertices, and an arc for every rectangle. It does capture incidence of rectangles to line segments, but adjacency between rectangles is not exactly covered. Consequently, the *grason h-digraph* is a subgraph of the line graph of the *polar h-digraph*.

The relations between the graphs acting as floorplans for the same rectangle dissection are brought together in the table of figure 4 where "D" stands for dualization and "L" for forming the line digraph. The upward arrow towards the polar digraphs and superset signs indicate the loss of information when going to polar graphs. Polar graphs are nevertheless the most popular graph models because they can be transformed into sets of linear equalities that must be satisfied by the rectangle dimensions of any compatible dissection[1]. This forms a good starting point for sizing using mathematical programming.

2.2 Trees

Trees for representing rectangle configurations have become popular recently. Soon we will see the appearance of *O-trees*, *bi-star trees* and *stair trees*. These representations compete in their efficiency to solve the packing problem for their compatible rectangle configurations. These packing problems are easy to solve for rectangles with fixed shape and orientation. An example is the *labeled ordered tree* [19] from which a packing can be computed in linear time. Among all possible trees with adequate number of vertices there is one that produces the smallest possible packing. There are however very many possible trees though their number is bounded by the *catalan number* times the number of permutations, and this number is asymptotically smaller than $(n!)^2$, the number of sequence pairs.

[1] In fact, these equalities are the kirchhoff equations with e.g. the widths as current intensities, and the heights as voltage differences between the nodes.

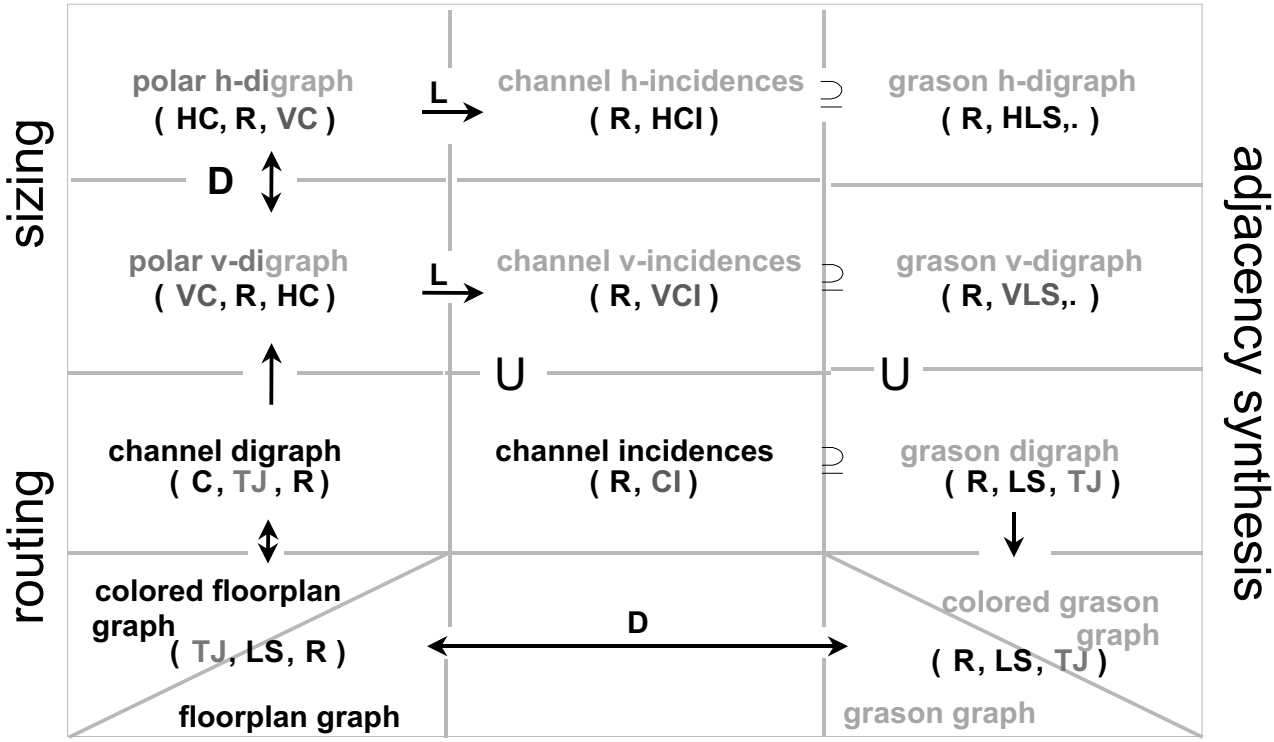

Figure 4: Relation table

2.3 Point configurations

Plane point sets are by now the most commonly used floorplans. If the only information that is used is the sequence of the points (representing the modules) after projecting them on an axis, then it is better to speak of a sequence pair, because all distance information is left unused.

Methods that try to embed a distance space into a euclidean space, and finally into a two dimensional euclidean space often use eigenvalues to find the plane with maximum spread and to loose as little as possible distance information when projecting on that plane. Depending on the distance metric certain statements about optimality can be made. However, the euclidean distance is not really a useful metric and projecting may force modules apart that should and can be kept together. Eigenvalue methods are mainly good for recovering structure in low dimensional configurations.

With the rise of annealing as a competitive method for improving complex configurations it was soon considered for floorplan design [16]. The question was how to represent relative positions in such a way that total wiring length was calculated very quickly. The first answer was a pair of sequences, which meant that the moves were transpositions of two elements in one such a sequence, but the points were then spaced according to their area (**not** the square root of area).

Sequence pairs got a more specific meaning when the two permutations of modules implied the following meaning for relative positions:

- a module m_i is to the left of module m_j if it precedes that module in both sequences
- a module m_i is above a module m_j if it precedes that module in the first sequence, and if it follows that module inde second sequence

Up to now sequence pairs are only generated by stochastic algorithms such as annealing. Evaluations are time consuming, albeit polynomial or even linear, which causes long runtimes, because these evaluations are placed in the inner loop. This makes such algorithms not very practical. In favor of sequence pairs it should be noted that they can easily handle empty rooms without explicitly encoding them in the sequences. This not possible for example with polar graphs!

3. FLOORPLAN DESIGN

In many ways one can argue that floorplan design has not been solved adequately. The various algorithms for generating floorplans modeled by graphs are either confined to subspaces that do not have to contain optimum or even near-optimum plans, or produce in essence a number of configurations that cannot be bound by a polynomial in the number of modules.

3.1 Adjacency-based floorplan design

The first thoughts on floorplan design were directed towards realizing the required or desired adjacencies. These adjacencies form a graph, and if this graph is a grason graph, a rectangle dissection with these adjacencies exists. Linear-

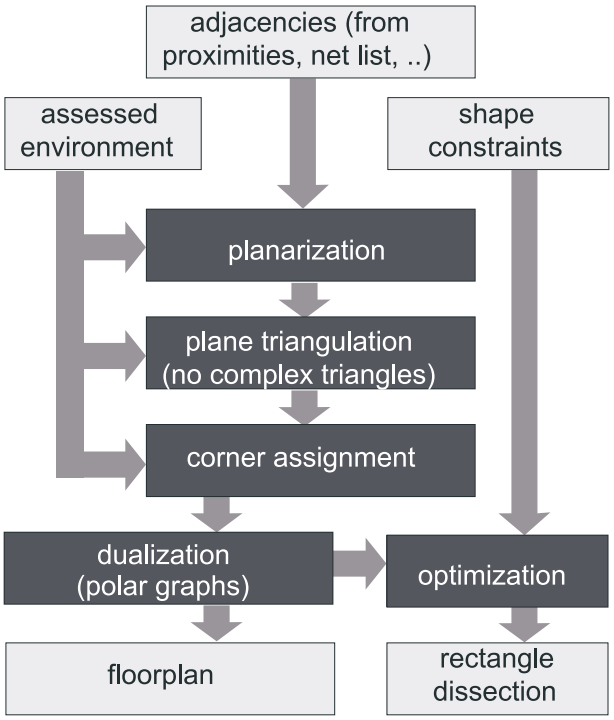

Figure 5: Adjacency-based floorplan design

time algorithms[2] for testing graphs on this property exist. In general, these adjacencies do not form a grason graph, and seldom a subgraph of a grason graph. The adjacency wishes therefore have to be pruned to such a subgraph and subsequently extended to a suitable grason graph, either explicitely or implicitely. A generic procedure is given in figure 5.

Graphs with rectangular duals (even graphs with duals, even when combinatorially defined) have to be planar. Procedures that try to realize adjacencies therefore often start with heuristics for planarizing the corresponding graph. A straightforward way for doing that is by removing "weak" adjacencies. Removing enough edges will planarize a graph sooner or later. But also adding nodes can have this effect. These "phony" rectangles (because nodes are supposed to represent rectangles) are called wiring nodes in [7]. Other heuristics can be found, but after the first stage in figure 5 the graph is supposed to be planar, so it has a dual and of course a plane representation. But does it have a rectangular dual? Not for a representation with *complex triangles* that is circuits of three edges with other edges inside. If that is the case there are a number of options:

- modify the representation
- remove the complex triangles
- triangulate avoiding complex triangles

In [20] all options are exploited albeit in the reverse sequence. Remaining freedom is used for assigning the corners.

[2]Do not trust the textbooks in this! Consult the original articles.

The resulting graph has a rectangular dual, and therefore a corresponding rectangle dissection can be found. A constructive procedure for generating such a dissection in an efficient way was published in [1]. An elegant method, based on the observation that construction is in essence assigning T-junctions, was *complete matching* in [10].

3.2 Proximity-based floorplan design

Point configurations can be generated efficiently depending on the objective. Eigenvalue methods are very fast but are bound to use a euclidean metric. Besides, they usually use projection to lower the dimensionality of the solution which may reduce distances leaving other modules unnecessarily far apart. Repeated probing based on eigenvalues makes the generation slow.

Probabilistic techniques, such as annealing, genetic algorithms and evolution, need very fast evaluations, which makes them resort to crude measures. For fixed rectangles techniques as in section 2.2 can be of advantage, but they cannot handle more flexible shape constraints.

Currently the best results seem to be obtained with analytic optimizers of the quadratically convergent newton-type, followed by legalizers. They are usually applied to a flat designs, that is without any hierarchical structure, regardless of the number of objects to be treated. Since these analytic optimizers do not obtain non-overlapping configurations of the objects to be allocated, they are classical examples of floorplanners, although literature often calls them placement procedures, which is only justified when combined with a valid legalizer

In any way, crucial for floorplanners is the metric by which the results are evaluated. In a companion paper Andrew Kahng will address the issue.

4. CONCLUDING REMARKS

Much more can be said about floorplanning, especially about the misconceptions and misinterpretations surrounding it. A particularly topic that attracts many scientist (but few tool developers) is the avoidance of special constraints. Early in its history the restraint of preserving sliceability was recommended. It was exactly that constraint that made floorplanning fit as a glove for stepwise refinement from a given hierarchy. For floorplans with that property could be represented as a tree. But there were other and more important properties. The fact that slicing structures have the minimum number of "channels" may have had his day, as well as the conflict free routing sequence of those floorplans. Or that the best slicing tree for routing can be found efficiently for a given sliceable placement. But the fact that many algorithms that are at least $\mathcal{NP}$-hard for general floorplan structures become tractable for slicing structures, remains of paramount importance. Among these is of course the floorplan optimization problem [14]. Later it was proven by [18] that even a simple special case of fixed geometry and free orientation was $\mathcal{NP}$-hard! Also, labeling a given tree to find the optimum slicing tree with that topology can be done in polynomial time [8]. And last but not least, given a point configuration (but actually a pair of sequences) the optimal compatible slicing structure can be found in polynomial time. "Compatible" means that every slice in the

floorplan corresponds exactly with rectangular set in the point configuration [4].

Considering the place and the nature of floorplanning, as capable of handling uncertainties, flexibility and preliminary positions, there is not much sense in paying any price for including also non-slicing floorplan topologies.

5. REFERENCES

[1] J. Bhasker and S. Sahni , "Representation and generation of rectangular dissections , "Proc. 23th Design Automation Conference Las Vegas(Nev) +USA , 1986, pp. 108-114

[2] J. Bhasker and S. Sahni , " A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph , "Networks, 17, 307-317 (1987)

[3] U. Flemming , " Representation and generation of rectangular dissections , "*Proc. 15th Design Automation Conference* Las Vegas (Nev), U.S.A. 1978, pp 138-144

[4] L.P.P.P. van Ginneken, R.H.J.M. Otten , " Optimal slicing of plane point placements , "*Proceeding of the European Design Automation Conference*, Glasgow, 1990, pp322-326

[5] J. Grason , " A dual Linear graph representation for space-filling location problems of the floor-planning type , " *MIT Press, Cambirdge (Mass),* , USA, 1970

[6] T. Günther , " Die räumliche Anordnung von Einheiten mit Wechselbeziehungen , " *Elektron. Daten Verarb. 6, 209-212* 1969

[7] W.R. Heller and G. Sorkin, K. Maling , "The planar package planner for system designers , "Proc. 19th Design Automation Conference Las Vegas (Nev) USA, 1982 pp. 663-670

[8] W. Keller, F. Schreiner, B. Schurman, E. Siepmann and G. Zimmerman , " Hierarchisches top down chip planning , " *Report University of Kaiserslautern* West Germany, 1987

[9] K. Kozminsky and E. Kinnen , " Rectangular duals of planar graphs , " *Networks*, 15, 145-157 (1985)

[10] S.M. Leinwand and Yen-Tai Lai , " An algorithm for building rectangular floor-plans , " *Proc. 21st Design Automation Conference* Albuquerque (NM) USA, 1984, pp. 663-664

[11] W.J. Mitchell, J.P. Steadman and R.S. Liggett , "Synthesis and optimization of small rectangular floor plans , "Environment and Planning B 3, 37-70 (1976)

[12] T. Ohtsuki, N. Sugiyama and H. Kawanishi , "An optimization technique for integrated circuit layout design , "Proc. ICCST Kyoto, Japan, 1970, pp. 67-68

[13] R.H.J.M. Otten , " Automatic floorplan design , " *Proc. 19th Design Automation Conference* Las Vegas (Nev.), U.S.A. 1982, pp 261-267

[14] R.H.J.M. Otten and P.S. Wolfe , "Algorithm for wiring space assignment in slicing floorplans , "Technical Report Y0882-0682 Thomas J. Watson Research Center, Yorktown Heights (NY) USA, 1982, IBM Technical Disclosure

[15] R.H.J.M. Otten , "Efficient floorplan optimization , "Proc. ICCD port Chester (NY) USA, 1983, pp. 499-502

[16] R.H.J.M. Otten, L.P.P.P. van Ginneken, , "Floorplan design using annealing , "Digest ICCAD, Santa Clara 1984

[17] R. Schmid , " Synthese von Floorplans auf der Basis von Dualgraphen mit rechtwinkliger Einbettung , " Ph.D. Thesis Universität Karlsruhe, West Germany, 1987

[18] L.Stockmeyer , "Optimal orientations of cells in slicing floorplan designs , "Information and control 57, 91-101 (1983)

[19] T. Takahashi , "A new encoding scheme for rectangle packing problem , "Proceedings ASP-DAC2000, Yokohama, Japan, 2000, pp175-178

[20] S. Tsukiyama, K. Koike and I. Shirakawa , "An algorithm to eliminate alle complex triangles in a maximal planar graph for use in vlsi floorplan , "Proc. ISCAS Philadelhia (Penn) USA, 1986, pp. 321-324

[21] K. Zibert adn R. Saal , "On computer aided hybrid circuit layout , "Proc ISCAS San Francisco, USA, 1974, pp. 314-318