Implementation of VLSI Routing Algorithms

Bence Golda, Bálint Laczay, Zoltán Ádám Mann, Csaba Megyeri, András Recski

Department of Computer Science
Budapest University of Technology and Economics
H-1117 Budapest, Magyar tudósok körútja 2
Hungary
vlsi@cs.bme.hu
http://www.cs.bme.hu/vlsi

This paper was accepted for publication in the Proceedings of the
IEEE 6th International Conference on Intelligent Engineering Systems, Opatija, 2002

Abstract— **In this paper we introduce a flexible software framework for the easy implementation and evaluation of VLSI routing algorithms and the visualization of routing results.**

Moreover, we present some new results in the theory of VLSI routing algorithms for rectangular grids, as well as the reformulation of some classical algorithms for triangular grids.

I. Introduction

In the last decades, with the amazing spread of electrical devices, the need for Very Large Scale Integrated (VLSI) circuits has grown rapidly. This required the construction of new algorithms and methodologies in all phases of VLSI circuit design and realization [1], [2], [3], [4]. As a result, a variety of routing algorithms, mathematical models, heuristics and automation tools have been developed [5], [6], [7], [8], [9].

In this paper, we consider the problem of *detailed routing*. That is, the placement of the units is already known, as well as the terminals to be interconnected. The aim is to determine if such an interconnection is possible, and in the case of a positive answer, an optimal interconnection is to be found [10], [11]. Optimality is usually measured in terms of chip size, but other factors (such as wire length, number of bends, speed, energy consumption etc.) can also be considered.

As the complexity of the circuits to be designed has grown significantly, it has become increasingly difficult to create suitable algorithms. First, the algorithms have to be fast and scalable in order to cope with huge and complex problem instances. A second problem is that because of the complexity of the routings—especially in the case of 3-dimensional routing—it is increasingly hard for the algorithm designer to visualize the operation of his or her algorithm. There has been little tool support for this [12].

One of the main contributions of the presented work is a flexible software framework, which enables rapid prototyping, evaluation and testing of routing algorithms. An important feature of the software is its support for both 2-dimensional and 3-dimensional visualization of routings.

We also implemented some of the 'classic' routing algorithms using the software. This enabled us to perform certain experiments and measurements, which gave interesting results about the practical effectiveness of the algorithms.

Another contribution of the paper is connected with a less wide-spread routing model based on a triangular grid instead of a rectangular one. We managed to transfer some known routing algorithms to this new model, yielding in some cases better results than in the original rectangular model. The software could also be easily extended to support the new model, which proved its flexibility.

The paper is organized as follows. Section II gives an overview of the formal mathematic model of detailed routing, as well as the main results in the field. This section also includes two previously unpublished results. Section III introduces the triangular model and describes routing algorithms for it. Section IV presents the software framework, while Section V describes some empirical results. Section VI concludes the paper.

II. VLSI routing theory

A. Basic definitions

In VLSI routing problems we are given a rectangular area and we have to interconnect some points of the boundary by wires. For this purpose we use a square grid or usually a three dimensional cubic grid containing l levels of square grids. These levels are called *layers*. The points at the bounds of the grid connecting to the devices are called *terminals*. A set of terminals to be interconnected is called a *net*. A *routing problem* is a collection of nets. To solve this problem we have to find a collection of vertex-disjoint connected subgraphs of the grid so that the terminals belonging to the same net are in the same subgraph. In the multilayer model, where $l \geq 2$, a wire can leave a layer for another using a *via-hole* and the subgraphs may leave the terminals at any layer, so via-holes are at the boundary as well.

If terminals appear only at one side of the boundary, we speak about a *single row routing problem*. If they arise on two parallel sides then it is a *channel routing problem*. If terminals are on all the four sides, the problem is called *switchbox routing*. Beside these important cases it is possible that terminals are on two adjacent sides what we call *gamma routing* or on three sides when we speak about *open*

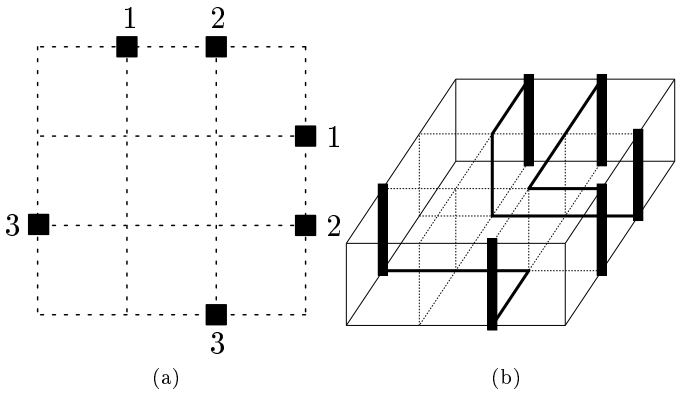

Fig. 1. A routing problem and its solution

switchbox routing.

In case of a single row routing or a channel routing the width is not given and we have to minimize the width of the solution, while in case of switchbox routings and gamma routings the size of the area is determined so we have to decide whether a solution exists. In this case an optimization problem can be to minimize the number of layers. Although we defined a solution as a collection of vertex-disjoint subgraphs, some authors studied solutions where the subgraphs have to be edge-disjoint. This is called *edge-disjoint routing.* If $l = 1$, we speak about *planar routing.* If there are several layers, we can consider certain models where restrictions are introduced. The most frequently used model is called *Manhattan model* where consecutive layers may contain wire segments of different directions (horizontal or vertical) only. If no restriction is defined, the model is called *unconstrained.*

Figure 1 shows an example switchbox routing problem (Figure 1(a)) and its solution in the two-layer unconstrained model (Figure 1(b)).

B. Algorithms and theoretical bounds

In this section we mention some results in different models. At first we consider the single row routing problem. In case of $l = 1$, one can easily verify that not every problem can be solved. If $l \geq 2$ then every specification can be solved in the Manhattan model by Gallai's algorithm based on a greedy interval packing. Similarly, single row routing problem can be solved in case of $l \geq 3$. The width of the solution is $\lceil \frac{d}{l_H} \rceil$, where d is the density and l_H is the number of horizontal layers. In Manhattan model this is the best possible. The theoretical lower bound in the unconstrained model is $\lceil \frac{d}{l} \rceil$, but this can slightly be improved:

Theorem 1: For every l and $w > 4$ there exists a routing specification with density $d = lw-1$ which cannot be solved in w width on l layers. In other words, the theoretical bound $\lceil \frac{d}{l} \rceil$ is not tight.

Proof: One can immediately see that not every specification can be solved if $d = lw$, since at a position where the density is equal to d, all rows are occupied by a wire of a net and only the wires next to the terminals' side can be reached. Hence for example, the routing problem $12 \ldots d12 \ldots d12 \ldots d$ remains unsolvable.

Even in case of $d = lw - 1$ there must be an unsolvable problem. Suppose the problem has d nets and starts and ends with $12 \ldots d$. Consider the problem only between these parts, let us say, of n terminals. We have only one free row and at one step at most the four neighboring wires can leave to this row, and at most l wires can be reached from the terminal side. This means that the number of different routing realizations is at most $(4l)^n$. The number of all routing problems is d^n and since $d = lw - 1 > 4l$, there must be unsolvable problems. ■

Now let us consider the more interesting channel routing problem. First we mention that it is NP-complete to decide whether a channel routing problem can be solved in the Manhattan model on two layers [8]. But we can solve it in the unconstrained model by the algorithm of A. Recski and F. Strzyzewski [13]. A disadvantage of this algorithm is that the width of the channel can be $\frac{3}{2}n$, though this upper bound can be improved with some heuristics.

In case of channel routing in the multilayer Manhattan model the theoretical lower bound is $\lceil \frac{d}{l_H} \rceil$. The greedy interval packing algorithm solves the problem width $w = \lceil \frac{d}{l_V - 1} \rceil$, where l_V is the number of vertical layers. For odd l in the VHV... V model the two bounds are the same. For even l it is NP-complete to decide whether a solution with $w = \lceil \frac{d}{l_H} \rceil$ exists in the multilayer Manhattan model with l_H horizontal layers.

Now consider the switchbox routing. Here we suppose that terminals cannot appear at the corner of the switchbox. There is no fixed number of layers that is sufficient for every specification, see an example of S. E. Hambrusch in [9]. This example shows that the number of layers depends on the shape of the rectangle. Szeszlér's algorithm [14] solves the switchbox routing problem in linear time in the Manhattan model on $2\lceil m \rceil + 4$ layers, where $m = max(\frac{h}{v}, \frac{v}{h})$, h and w are the height and width of the switchbox.

In the switchbox routing the basic lower bound of the number of layers, obtained by the example of Hambrusch, is $\lceil m \rceil + 1$ in the unconstrained model and $2\lceil m \rceil + 2$ in the Manhattan model. Due to the algorithm of D. Szeszlér, we have an upper bound of $2\lceil m \rceil + 4$ in both cases. For square shape box in the Manhattan model this means $4 \leq l \leq 6$.

Theorem 2: The construction of Hambrusch with size $w = h = 2$ (see [9]) cannot be solved on 5 layers. This means that the lower bound for square boxes in the Manhattan model is 6.

Proof: Note that it is still possible that for larger boxes less than 6 layers are sufficient. But in this case, every net uses at least 5 vertices of the grid because the shortest path between the endpoints is of length 3, but every wire has to turn at least twice and in Manhattan model this needs a via hole. The four nets use at least 20 vertices, thus the number of layers is at least five.

In case of five layers every vertex is occupied. We can suppose that there are two layers for horizontal wires. Then the horizontal wires fill these layers. At the top layer only the Southern or Northern terminals can connect to the grid bacause this layer is for vertical wires. But the wires on the

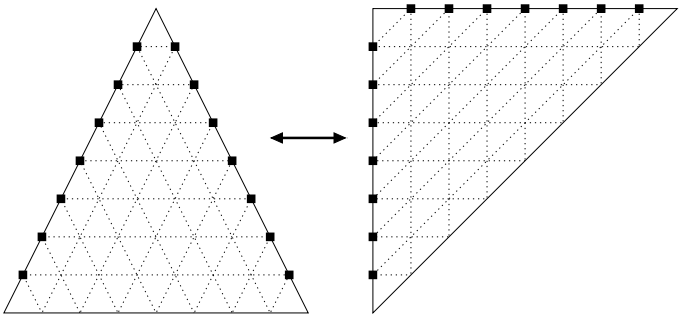

Fig. 2. The connection between lambda and gamma routing

second layer cut these terminals from the wires of the fourth layer, hence the wires on the second layer are for the Southern and Northern terminals. This can be repeated for the wires of the fourth layer, so Western and Eastern terminals do not have horizontal wires, leading to a contradiction. ■

Like channel routing, the two-layer switchbox routing in the Manhattan model is also NP-hard.

III. The triangular routing model

VLSI routings are usually based on the square lattice. It is an interesting question whether other type of lattices like the triangular or the hexagonal lattice are also useful for solving routing problems. Since on these lattices we can use one more direction for wires, one can expect that the number of necessary layers may decrease. In technology, the use of a new direction causes no problem, while via holes remain the same as before.

One can immediately see that the triangular lattice is a better candidate than the hexagonal lattice, because on the latter wires have to turn at every vertex of the lattice. This is the main reason why we focus on the triangular lattice.

In this case, it is natural to define the routing area as a triangle rather than a rectangle. Similarly to the original model, we call a problem *single row routing problem*, if terminals are only on one side of the triangle. If terminals appear on two sides, we speak about a *lambda routing problem* while the general case, when terminals can be everywhere, is called *triangle routing problem*.

Note that the lambda routing problem differs from the original gamma routing problem in the fact that it can use three directions instead of two on the half of the area (see Figure 2).

We call a routing *Manhattan type* if layers contain wires of the same direction and these directions change from layer to layer. Now we will show some algorithms on the triangular model. The sides of the triangle will be called South, East and West, while the nonhorizontal directions of the lattice will be called Eastern and Western. Terminals in single row problems will always occupy the South side, while in lambda routing the East and North side.

A. Single Row Routing

The single row routing can be solved in this model like in the original one:

Theorem 3: A single row routing can be solved on one layer in the triangular model if and only if it can be solved on one layer in the original model. Moreover, similarly to the original model, every single row routing can be solved on two layers in the triangular model. A Manhattan type solution can be realized in linear time.

Proof: A single row routing can be solved on one layer if and only if no two nets are crossing and this is independent of the type of the lattice. On two layers choose the leftmost terminal of every net. Give a wire segment from this terminal on the first layer in the Eastern direction and connect every other terminal on the second layer using a West-directed segment. ■

B. Lambda Routing

Theorem 4: Every lambda routing problem can be solved Manhattan-like on 5 layers in linear time.

Proof: The first and fifth layer will be for Eastern, the second and fourth layer will be for horizontal and the third layer will be for Western segments. Choose a net that has terminal on the West side and give a wire segment on the second and third layer from the lowest terminal of this net. Now every other terminal of the Western side can be connected on the third layer, while terminals on the Eastern side on either the first or on the fourth layer. Nets having Eastern terminals only, will be given a segment on the fifth layer from the topmost terminal. Other Eastern terminals can be connected on the fourth layer. ■

C. Triangle Routing

Theorem 5: The number of layers needed in the triangle routing is at least 4, while at least 5 layers are needed for a Manhattan type solution. There is a linear time algorithm that uses 12 layers to provide a Manhattan type solution.

Proof: One can find small specifications needing 4 or 5 layers in the Manhattan type routing. By using the lambda routing algorithm we get a 12-layer-algorithm for triangle routing. We have to use the above algorithm to connect nets on the first four layers, having only Western terminals, and nets having both Western and Eastern terminals. We do not need the fifth layer. Now repeat this for nets having Eastern terminals only, and for nets having both Eastern and Southern terminals on the next four layers. Finally, do the same with nets having Southern terminals only, and for nets having both Western and Southern terminals on the last four layers. ■

D. Gamma Routing

In the original model, gamma routing can be solved by Szeszlér's switchbox algorithm on 4 layers. However, it is possible to solve the gamma routing problems on 3 layers, if we use the triangular lattice:

Theorem 6: Every square-shape gamma routing problem can be solved in linear time on 3 layers in Manhattan way on the triangular lattice.

Proof: Note that now we have a square box on the square lattice extended by a North-East diagonal direction, terminals are on the Northern and Western side.

The first layer is for vertical, the second is for horizontal and the third is for diagonal wires. Suppose that the square has size $n \times n$, that is, there are at most $2n$ terminals and n nets. We give a wire segment for every net on the second layer. This is possible because we have n rows and at most n nets. Nets having at least two terminals on the West side get a wire on the horizontal layer from their topmost terminal. Next the nets that have exactly one terminal on the West side get a wire on the horizontal layer from this western terminal. On the rest of the rows of the second layer we give one wire segment for every net that has Northern terminals only. Now every Northern terminal can be connected on the first layer and every Western terminal can be connected on the third layer.

This algorithm works also in the general rectangular case. Suppose that width is no greater than height, $w \leq h$. If we have N nets then $3\lceil \frac{N}{w} \rceil$ layers suffice, since horizontal tracks can be placed on $\lceil \frac{N}{w} \rceil$ layers. As $N \leq \frac{w+h}{2}$, the routing uses at most $3\lceil \frac{w+h}{2w} \rceil$ layers. That is at most $3m$ layers. ■

IV. The software framework

After having described briefly the theory of detailed routing algorithms as well as triangular routing, the second part of the paper focuses on more practical issues. This section presents our software implementation.

The aim was to create a software that helps in designing and evaluating routing algorithms. Correspondingly, the main design goals were the following:

- Portability;
- Both 2-dimensional and 3-dimensional visualization;
- The implementation of routing algorithms should be as easy as possible;
- Also heuristic algorithms have to be supported;
- Flexibility to incorporate new routing models;
- Persistent storage of problem instances and routings with a suitable file format, which is easy to generate and to process;
- Automatic generation of figures for presentation and educational purposes.

The software was implemented in Java, using object-oriented methodology, and UML as modelling language. For 3-dimensional visualization, Java3D was used. For the persistent storage of routings, we defined a special format using XML [15]. Routings can also be exported in FIG format, so that they can be edited using the drawing tool XFig, converted to PostScript and embedded in LaTeXdocuments.

The software was mainly tested on Pentium PIII workstations running Debian Linux. It can be downloaded from the website of the project. It needs JDK 1.3, Java3D and the XML parser JAXP.

The software consists of three major modules: the graphical user interface (GUI), the backend library, and the implemented algorithms. Figure 3 shows a simplified UML diagram of the system.

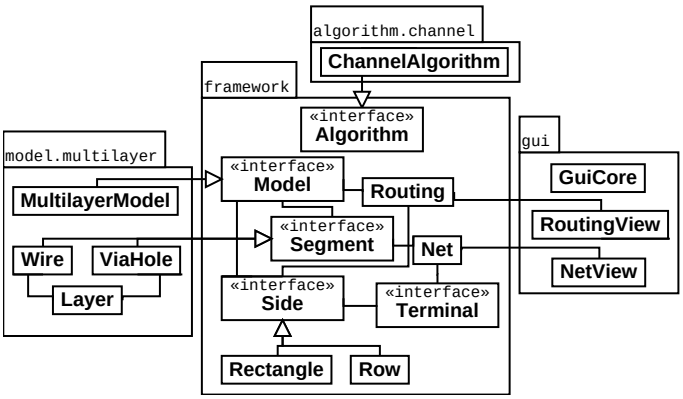

Fig. 3. Simplified UML diagram of the software

A. The backend library

The backend library is a set of classes modelling the important elements of routing problems and thus supporting the implementation of routing algorithms. These classes include:

- `Net`
- `Wire`
- `ViaHole`
- `Routing`
- etc.

Other elements of the routing problems are modelled as *interfaces*, the implementation of which depends on the actual routing model:

- `Terminal`
- `Side`
- etc.

The most important principle in the design of the backend library was flexibility. In particular, the geometric properties of the routing problem (*i.e.* the routing model) are separated from the logic of the routing problem (interconnection of terminals belonging to the same net by wires). This way, new routing models can be defined easily; they only have to implement certain interfaces. The flexibility of this approach has been proven by the uncomplicated implementation of the triangular routing model *after* the development of the system was already completed.

Currently, two routing models are implemented: the traditional rectangular grid (`MultiLayerModel`) and the new triangular grid (`TriangularModel`). Additional models can also be realized; however, each model needs a corresponding graphical user interface.

B. Algorithms

To demonstrate the operation of the system, we implemented the following algorithms (all of them are realized as separate classes implementing the `Algorithm` interface):

- Gallai's algorithm for the single row routing problem [10, p. 267];
- The algorithm of Recski and Strzyzewski for the channel routing problem [13];
- Szeszlér's algorithm for the switchbox routing problem [14].

The implementation of the algorithms was significantly simplified by the services of the backend library. On the other hand, we also experienced problems concerning the programming language. Although Java has proven to be really suitable for constructing the backend library, it may not be the best choice for implementing algorithms. It seems to be hard to maintain flexibility in the implementation of the algorithms in Java.

The implementation of Gallai's algorithm revealed an interesting fact: although the literature states that it is a linear-time algorithm, this is not true for its description in most sources. So extra care had to be taken to implement it in a more efficient way.

C. GUI

The graphical user interface is mostly made up of standard Swing elements. The key design patterns used in the GUI are *model-view* and *listener* [16]. On the other hand, the design of the user interface also required the application of more user-focused principles such as *similarity* and *symmetry*.

The main control window contains two lists: one for algorithms and one for routings. Both lists can be edited by adding and removing elements. The user can select an algorithm and a corresponding routing problem and let the algorithm solve the given problem. View windows (2D and 3D) showing the solution automatically show up.

Creating a new routing problem is supported by a 'wizard'. The user can first either define a set of constraints or use a predefined one and fill in its preferences (*e.g.* the length in the case of a single row routing problem). After that, terminals and nets can be added to the routing problem in a special window. Alternatively, a previously saved problem instance can be loaded from the disk.

V. Empirical results

The implementation of the routing algorithms described in Section IV-B enabled us to test their runtime properties. The main purpose of the testing methods in this section is the evaluation of heuristic algorithms; however, they also yielded interesting results in the case of the implemented exact and deterministic algorithms. Even if these algorithms are claimed to be linear-time in the literature, we saw in Section IV-B that it may still be problematic to construct a linear-time implementation. Moreover, it is important to know the constants of the linear implementation, as well as the impact of technical influences, such as memory management, on the running time.

A. Functional tests

The aim of the first tests was to check the correctness of the implementation. For this purpose, we chose problem instances that were small enough to verify the operation of the algorithms manually. These test cases also covered special cases, such as empty nets, no nets at all etc. The verification process consisted of three steps:

1. We checked first whether the algorithm ran without errors, and terminated;
2. It was checked whether the output of the algorithm was a correct solution of the original problem;
3. The algorithm was also performed manually, and the results were compared.

Besides finding a couple of minor bugs, which were fixed right away, an interesting behaviour of the checkbox routing algorithm was recorded. Namely, in some cases the algorithm produces routings with loops. This does not affect the correctness of the output; however, in an industrial application, such loops should be eliminated.

B. Performance tests

In the performance tests, the average running time of the algorithms was measured as a function of problem size. The problem size was characterized by the number of terminal positions. The test cases were generated randomly, based on two different distributions. In the first case the number of terminals in a given net was determined by a binomial distribution, in the second case this number was constant for all nets. In both cases the terminals were selected according to a uniform distribution.

Since we wanted to measure the performance of the algorithms, extra care had to be taken to exclude the starting time of the Java virtual machine as well as the time required by test case generation and garbage collection from the measurements. Particularly, garbage collection has to be forced before starting the tests to avoid compromising the measurements with the deferred clean-up of memory that had been reserved before the algorithm was started.

Moreover, in the case of big problem instances, the first invocation of the algorithm was significantly slowed down by the fact that the Java virtual machine had to reserve huge memory blocks from the operating system, while the second invocation already started with more memory available. So, we let the algorithm solve a fix-sized problem before each measurement.

We found that the two considered distributions gave very similar results. Therefore we mostly focused only on the case when the number of terminals in a net is constant. The case when this constant is 2 is particularly important, because this is the situation in many practical applications.

In the case of Gallai's algorithm for the single row routing problem, we found that for small and medium size problems the running time of the algorithm is indeed linear, with a slant of about $0.01 \ldots 0.02$ ms/terminal. However, on very large problem instances ($10^4 \ldots 10^5$ terminals, which is not typical for detailed routing problems in practice) it was superlinear with an exponent of about 1.22.

In the case of the channel routing algorithm, the slant for normal problems was in the range $0.026 \ldots 0.036$, and the exponent for huge problem instances was about 1.08. For the switchbox routing algorithm, the slant for normal-size problems was $0.016 \ldots 0.024$, and the exponent for very large problem instances was about 1.13.

The superlinear behaviour of the algorithms in the range of huge problem instances is probably caused by the effects of virtual memory management. Operations that are assumed to require constant time (such as, for instance,

referencing an element of an array) require an increasing amount of time if the amount of data becomes large because of the page and segment access overhead. This also explains why the level of superlinearity was the highest in the case of Gallai's algorithm, since it makes heavy use of memory for optimization.

VI. Conclusion

This paper has presented both theoretical and practical results in the field of VLSI detailed routing algorithms.

As theoretical contributions, we gave two new theorems which improve some lower bounds on the number of necessary layers: we have proven that the previously known lower bound for the number of necessary layers to solve the single row routing problem in the unconstrained model is not tight; and that the lower bound for square-shaped switchbox routing problems in Manhattan model is 6 instead of 5.

Moreover, we investigated a less widespread model, the triangular model, and managed to transfer some algorithms for it. In some cases these algorithms yield even better results in the triangular model than in the rectangular one.

As a practical contribution, we presented our software framework for the easy implementation and evaluation of routing algorithms. This framework was flexible enough to enable the implementation of the triangular model as well. Also, the software posesses excellent visualization capabilities.

Finally, we performed a series of tests on the implemented routing algorithms. Both the functional and the non-functional tests gave interesting results that could not have been discovered without the help of the presented software.

Our future research plans include the implementation of additional algorithms (including algorithms for the triangular grid and also heuristic algorithms). Moreover, we would like to extend the software with high-level debugging functionality.

VII. Acknowledgements

The work of the group was partially supported by grant No. OTKA 30122 of the Hungarian National Science Fund and that of Zoltán Ádám Mann by a grant by Timber Hill LLC.

References

[1] R. N. Noyce, "Microelectronics," *Sci. Amer.*, vol. 237, 1977.

[2] T. C. Hu and Ernest S. Kuh, *VLSI Circuit Layout: Theory and Design*, chapter "Theory and Concepts of Circuit Layout", pp. 3–19, IEEE Press, New York, NY 10017, 1985.

[3] Otto G. Folberth and Warren D. Grobman, Eds., *VLSI: Technology and Design*, chapter "Device, Circuit, and Technology Scaling to Micron and Submicron Dimensions", pp. 3–18, IEEE Press, New York, NY 10017, 1984.

[4] Otto G. Folberth and Warren D. Grobman, Eds., *VLSI: Technology and Design*, chapter "VLSI Design Automation: An Introduction", pp. 19–23, IEEE Press, New York, NY 10017, 1984.

[5] A. Frank, "Disjoint paths in a rectilinear grid," *Combinatorica*, vol. 2, pp. 361–371, 1982.

[6] M. L. Brady and D. J. Brown, "VLSI routing: Four layer's suffice," *Advances in Computing Research*, vol. 2, pp. 245–258, 1984.

[7] M. Koebe and P. Dupont, "Single-layer channel routing," *J. Inf. Process. Cybern.*, vol. 7/8, pp. 339–354, 1988.

[8] A. S. LaPaugh, "A polynomial time algorithm for optimal routing around a rectangle," *Proc. 21st FOCS Symp.*, pp. 282–293, 1980.

[9] S. E. Hambrusch, *Channel routing in overlap models*, IEEE Trans. Computer-Aided Design of Integrated Circ. Syst. CAD-4, 1985.

[10] A. Recski, "Minimax results and polynomial algorithms in VLSI routing," *Ann. Discrete Math*, pp. 261–273, 1992.

[11] D. Szeszlér, "Nagy bonyolultságú hálózatok huzalozása," Kézirat, 2000.

[12] Dorothea Wagner and Karsten Weihe, "An animated library of combinatorial VLSI-routing algorithms," in *Symposium on Computational Geometry*, 1995, pp. C28–C29.

[13] Recski A. and F. Strzyzewski, "Vertex-disjoint channel routing on two layers," in *Integer programming and combinatorial optimization*. 1990, pp. 397–405, University Waterloo Press.

[14] D. Szeszlér, "Switchbox routing in the multilayer manhattan model," *Ann. Univ. Sci. Budapest Eötvös Sect. Math.*, vol. 40, pp. 155–164, 1997.

[15] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, *Extensible Markup Language (XML) 1.0 Recommendation*, World Wide Web Consortium, second edition, 2000.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Massachusetts, 1994.

[17] *Java Virtual Machine Profiler Interface*, Sun Microsystems, 1999.