Lehrstuhl für Entwurfsautomatisierung
der Technischen Universität München

Efficient Quadratic Placement of VLSI Circuits

Peter Spindler

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. techn. Josef A. Nossek

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. Frank M. Johannes
2. Univ.-Prof. Dr.-Ing. Jens Lienig
   Technische Universität Dresden

Die Dissertation wurde am 20.12.2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 05.06.2008 angenommen.

Acknowledgment

This thesis is the result of my work and all the help and advice that has been provided to me through all of my years at the Institute for Electronic Design Automation, Technische Universität München, Germany.

The person who deserves most of my credit is undoubtedly my adviser Professor Frank M. Johannes. With the prolific discussions, not only about the field of research, and his continuous encouragement, he gave the essential requirements for successful research. I am also very grateful for the constant support and inspiration of Professor Ulf Schlichtmann. I would also like to thank Professor Jens Lienig and Professor Josef A. Nossek for their interest in my thesis and their job as reviewers.

The many interesting discussions with my colleagues contributed much to this thesis. Amongst others, I would like to thank Martin Strasser, Dr. Helmut Gräb, and my predecessor in the layout group, Dr. Bernd Obermeier. Special thanks go to Dr. Hans Eisenmann, who laid with his work the basis for this thesis. I owe respect to Dr. Bernd Finkbein, Getraude Kallweit, Hans Ranke, Werner Tolle, Susanne Werner, and Jürgen Zenz for their administrative and technical support.

Finally, I would like to thank with all my heart my girlfriend Katrin Mayer-Arnold, who always stands behind me and supports me under all circumstances.

Contents

Chapter 1

Introduction

Integrated circuits (ICs) are part of our daily live as they are the hearts of MP3 players, cell phones, personal digital assistants (PDAs), laptops, and even cars have a high number of integrated circuits. Also the industry mainly depends on integrated circuits in different applications, ranging from simulations of complex processes on main-frame computers to efficient control of production lines.

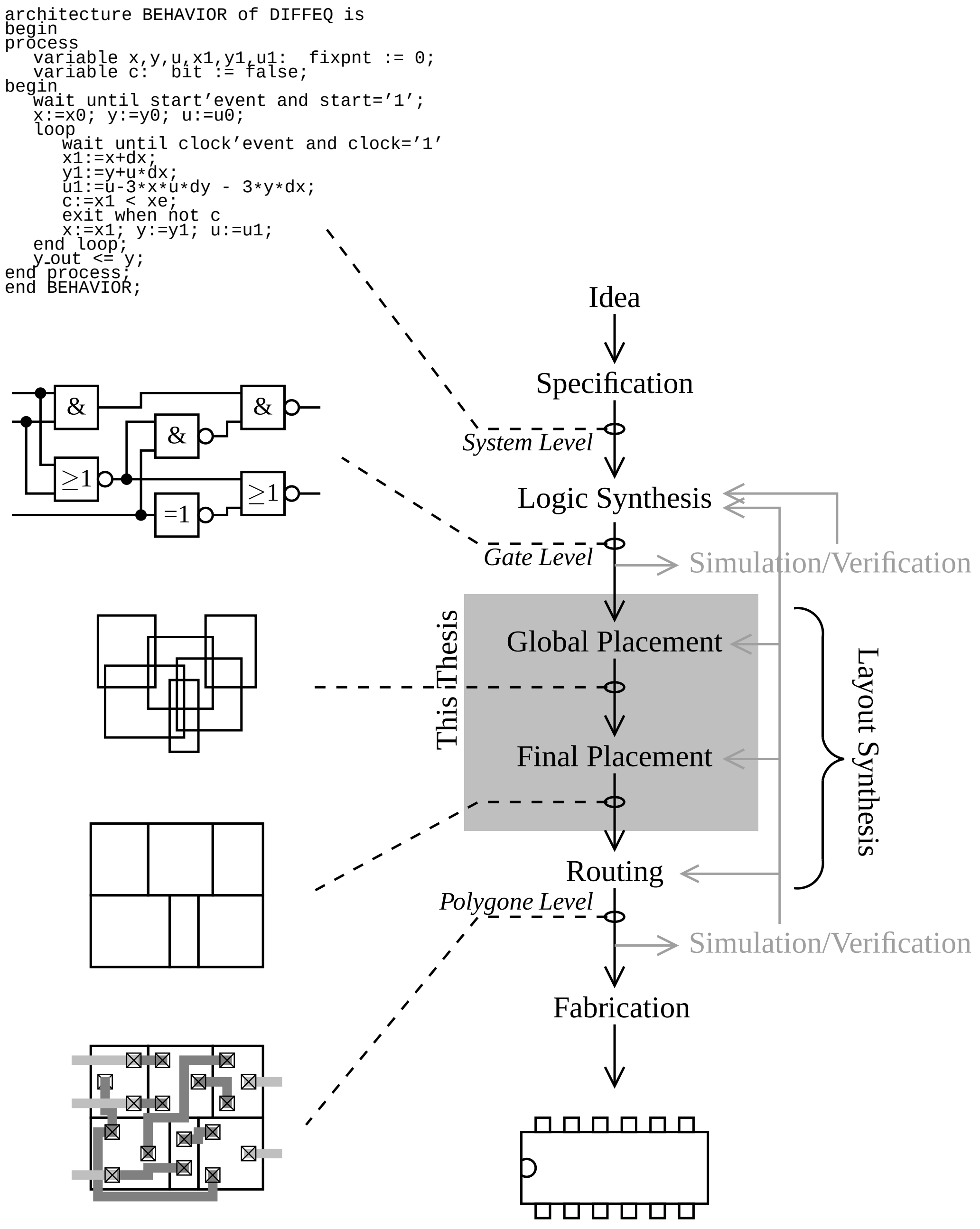
The history of integrated circuits started around 1960, when analog components were integrated on a piece of silicon for the first time. In 1971, Intel presented the 4004, the first microprocessor of the world with about 2300 transistors. At the time this thesis was written, integrated circuits can have billions of transistors. Hence, integrated circuits are today mostly called VLSI circuits, with VLSI standing for very large scale integration. This enormous complexity of integrated circuits can only be handled if the circuits are designed not by hand, but by algorithms, executed on computers. The usage of such computer algorithms in order to design integrated circuits is called electronic design automation (EDA).

In the year 1965, Gordon Moore [Moo65] detected that the numbers of transistors in an integrated circuit is doubling every 18 months (approximately). Still today, Moore's law is valid [SEM], which means that the complexity of integrated circuit is steadily growing. Therefore, fast and efficient algorithms are necessary for the EDA of future circuits.

1.1 Electronic Design Automation

Starting from the idea of a circuit, electronic design automation is done in several steps [SY95, Lie06], as shown in Figure 1.1. In each step, the description of the circuit is refined. After all steps, the circuit can be fabricated.

The first step of EDA is to specify the circuit. Here, the main features like performance, functionality, and physical dimensions are defined. Amongst others, also decisions on the architecture have be done, e.g., which type of processor, or which kind of memory the circuit should use. After this, the circuit is described as a behavior modeled at system level using a hardware description language like VHDL or Verilog. The next step is logic synthesis, which first transforms the behavior description of the circuit into a register transfer description. At register transfer level, the circuit mainly consists of a control unit and a data path. The data



```
architecture BEHAVIOR of DIFFEQ is
begin
process
   variable x,y,u,x1,y1,u1:  fixpnt := 0;
   variable c:  bit := false;
begin
   wait until start’event and start=’1’;
   x:=x0; y:=y0; u:=u0;
   loop
      wait until clock’event and clock=’1’
      x1:=x+dx;
      y1:=y+u*dx;
      u1:=u-3*x*u*dy - 3*y*dx;
      c:=x1 < xe;
      exit when not c
      x:=x1; y:=y1; u:=u1;
   end loop;
   y_out <= y;
end process;
end BEHAVIOR;
```

Figure 1.1: Design Flow of Integrated Circuits

path includes registers and functional blocks like arithmetic logic units. Moreover, the data are described as bit vectors. Based on the register transfer model of the circuit, the logic synthesis constructs the gate level description then. At gate level, the circuit consists of gates like inverters, and-gates, or-gates, flip-flops, etc. The gates themselves consists of transistors. The data are described as single bits. After logic synthesis, the gate level description of the circuit is simulated, and different specifications are verified, e.g., the maximal clock frequency. If the specifications are not met, the logic synthesis is done again. If the circuit is working correctly at gate-level, layout synthesis is done next. The main steps of layout synthesis is placement of the gates, and routing of the nets, which interconnect the gates. However, prior to placement, floorplanning is invoked to determine the positions of the I/O pins, the dimensions of big gates, and the dimensions of the chip. Due to the high numbers of gates, placement itself is done in two steps: global placement and final placement. During global placement, the gates are roughly spread on the chip. Final placement then removes the remaining overlap, aligns the gates to a given row/grid structure. There, different design rules are considered, like minimal distances between the gates. This thesis presents novel approaches for global and final placement. After the gates are placed, the nets, which interconnect the gates, are routed. After routing, the polygon level of the circuit is reached, i.e., the circuit is described only by polygons now. At polygon level, the circuit is simulated again, and it is checked if all given specifications are met. If not, the EDA is started from previous steps, and if necessary, it is even started again with logic synthesis. At the end of EDA, the lithography masks are created, and the circuit is fabricated using these masks.

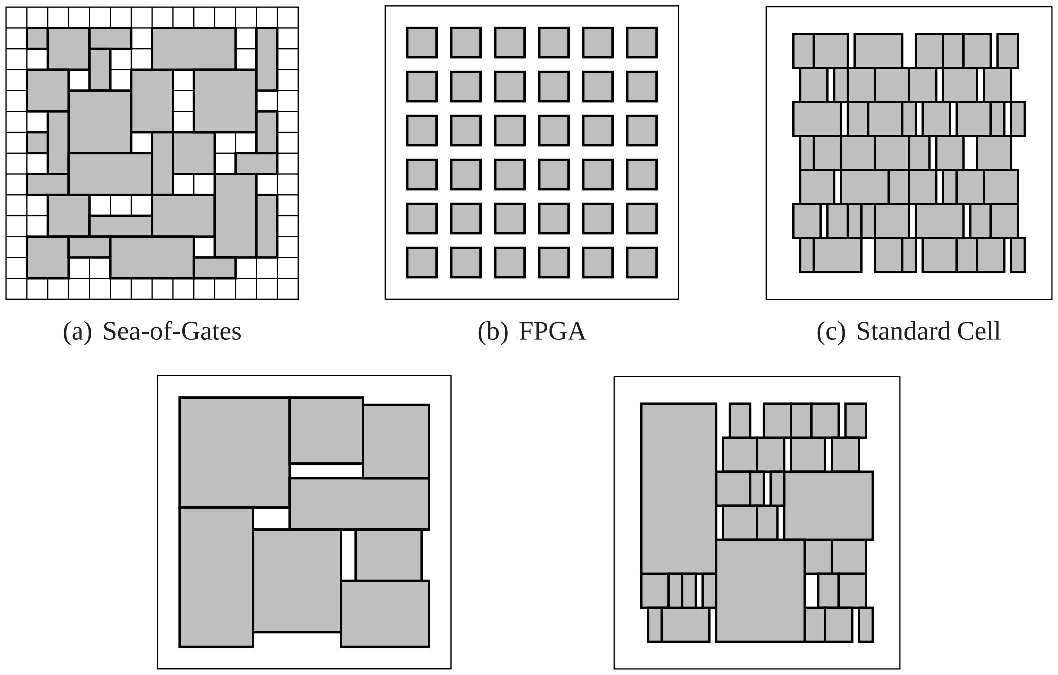
1.2 Types of Integrated Circuits

Figure 1.2 displays different types of integrated circuits used today. Each circuit type reflect one design style. The differences between them is mainly the type of gates, and how they are implemented on the “die”. “Die” here means the piece of silicon which implements the circuit.

1. Mask-Programmable Gate-Arrays/Sea-of-Gates
   The dies of mask-programmable gate-arrays and the dies of sea-of-gates have prefabricated transistors, aligned in a regular pattern. To implement circuits with such dies, the gates of the circuit are broken down to transistors first. Then, the gates as groups of transistors are assigned (placed) to the prefabricated transistors of the die. The routing is done in metal layers, either in channels between the transistors (mask-programmable gate-arrays), or above the transistors (sea-of-gates).

2. Field-Programmable Gate-Arrays (FPGA)
   The die of a FPGA is completely prefabricated, and consists of a regular matrix of programmable logic blocks and interconnect blocks. Placement of FPGAs means to assign gates of the circuit to the logic blocks of the FPGA. Routing is done by configuring the interconnect blocks.

3. Standard Cell Circuits
   The die of a standard cell circuit is not prefabricated. The circuit is implemented with



(a) Sea-of-Gates (b) FPGA (c) Standard Cell

(d) Macro Cell (e) Mixed-Size

Figure 1.2: Different circuit types

gates all having the same height but different widths. Such gates are called standard cells. Placement of standard cell circuits means to align the cells to a row structure. Today, routing of standard cell circuits is done mostly above the standard cells using various routing layers.

4. Macro Cell Circuits
   Similar to standard cell circuits, the dies of macro cell circuits are not prefabricated. Macro cell circuits consists of a few, but complex macro blocks, e.g., memory blocks, arithmetic units, or even processor cores. Today, these macros are often so called intellectual property (IP) cores. IP cores are purchased and are available at different description levels: system level (in VHDL or Verilog), at gate-level, or even at polygon level. Considering placement, there are two types of macros. Soft macros have a fixed area but are free in the aspect ratio (relation between width and height). Hard macros have fixed widths and heights. Therefore, placement of circuits with soft macros means not only determining the position of the macros, but also the aspect ratio.

5. Mixed-Size Circuits
   Mixed-size circuits consist of a few macros and a high number of standard cells. This circuit type is mostly used today.

Figure 1.3 shows two modern design styles based on state-of-the-art circuits: (a) mixed-size, and (b) macro cells. Due to the high number of standard cells, these cells are represented as “black clouds” around the gray macros in Figure 1.3 (a). The macro cell circuit depicted in Figure 1.3 (b) represents the widely used System-on-Chip (SoC) design style. There, each macro can represent one system, e.g., processor core, cache block, or network stack.

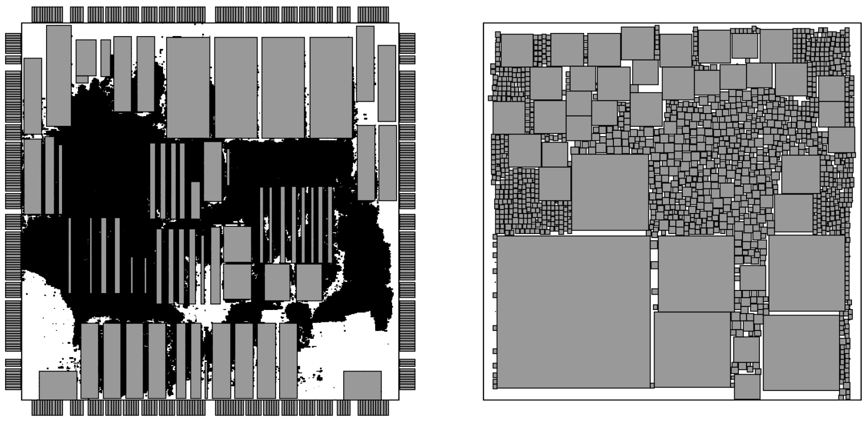

(a) Mixed-Size (b) Macros (SoC)

Figure 1.3: Two modern design styles.

1.3 Placement

Placement is one important step of the EDA flow (see Figure 1.1), which highly affects the quality of a circuit. The input of placement is the circuit described at gate-level. This means that the circuit consists of gates, and the gates are interconnected by nets. In the rest of the paper, the gates are called modules. Placement is to determine the positions of the modules, while considering different objectives and constraints. The fundamental constraints are that the modules do not overlap, and that all modules are located within the chip area. Here, it should be noted that today, the chip area is mostly given by floorplanning. An additional constraint of placement is for example to align the modules to rows or to a grid structure. The main objective of placement is to minimize the total wirelength, i.e., to minimize the sum of the lengths of all nets. This objective is used because with a minimal wirelength, the circuit is easy to route, the maximal clock frequency is high, and the power consumption is low. In summary, placement can be formulated as to solve the following problem.

Placement Problem:

Place all modules such that
all relevant objectives (e.g., total wirelength) are optimal and
all constraints (e.g., no overlap) are met.

Chapter 2

State of the Art

Although the placement problem proposed in the previous section sounds easy, it is a combinatorial problem, which is known to be a NP-complete problem [GJ79, Don80, SB80, Len88, Len90]. This means, there exists no algorithm up to date, which solves the problem optimal with polynomial runtime complexity. In the extreme case, all feasible placements have to be inspected, in order to find the optimal placement. With millions of modules (which is the number of modules in modern VLSI circuits), the number of feasible placements is quite high, i.e., the runtime is not practicable.

Hence, to get good solutions in polynomial runtime, the placement problem is solved by heuristics. One traditional method is to use two steps for placement: global and final placement. In global placement, the modules are spread roughly on the chip, with few overlap remaining. In final placement, the overlap is removed, and the modules are aligned to the grid/row structure. This thesis covers novel solutions for both placement steps. In the following, the state-of-the-art in global placement is described first, including different aspects as net models and routability optimization. Second, the state-of-the-art in final placement is presented.

2.1 Global Placement

Global placement means to spread the modules roughly on the chip, resulting in a placement with few overlaps. In the previous decades, different algorithms for global placement were developed. They differ mainly in the way how the wirelength is minimized, and how the modules are spread on the chip. Figure 2.1 categorizes different techniques, and lists the names of different state-of-the-art placers. Some of these techniques are able to spread the modules without any overlap on the chip. However, they are mostly stopped if there is only little overlap remaining. This overlap is removed in final placement then.

2.1.1 Greedy Placement

Placers based on greedy methods have in common to modify a given start placement over a sequence of iterations, and accept only better placements according to their cost. Here, the

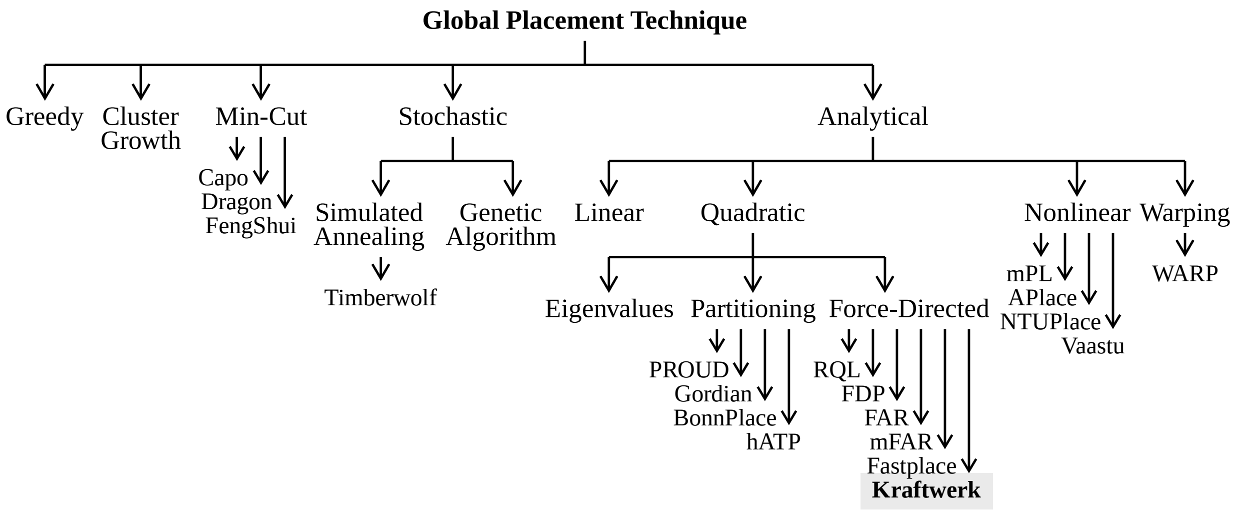


Figure 2.1: Different placement techniques and names of various placers.

start placement can be random, and the cost is mainly a combination of wirelength and overlap. Due to the fact that only better placements are accepted, greedy placers are likely to get stuck in a local minimum, i.e., they will probably not find the optimal solution. In principle, greedy placers modify the placement by permuting modules, either just two modules [HK72, Shu75, Sch76, Bla85a, Bla85b, CP80, IKB83, KP77, HWA76], or three and more modules [HWA76, Got79, Got81]. However, only for circuits with just a few modules, all possible modifications can be tested. For bigger circuits, only neighboring modules can be permuted in practicable runtime. Therefore, heuristics were developed to decide which modules are best to permute [Qui75, HWA76, Got79, Got81]. The main drawback of greedy placers is that they only do a local optimization of the placement. Thus, they highly depend on the start placement.

2.1.2 Cluster-Growth

Placers based on cluster-growth iteratively cluster new modules around already placed modules. Here, the first placed modules can be random. The strategy of cluster-growth placers can be viewed as bottom-up: starting from some placed modules, more and more modules are placed, until all modules are placed. The decision, which modules are clustered, is done based on a cost function representing the wirelength and the module overlap. Placers using this method are for example [SU72, HK72, Shu75, Sch76, HWA76, KP77, Got79, Got81, DK87, LM90, Mül90, YK92, KK92, Lee93, SSL93]. These approaches have good results for small circuits, but degrade with increasing numbers of modules per circuit. This problem is due to the local view of the method, and due to the high dependence on the start placement.

2.1.3 Min-Cut Placement

In contrast to the bottom-up strategy of cluster-growth placers, placers based on min-cut are following a top-down technique. Here, the placement area and the circuit are recursively partitioned. In doing so, parts of the circuit are assigned to parts of the placement area. The recursive process is done until each module is assigned to a unique part of the placement area, which results in a placement with no or just little overlap. The partitioning of the circuit is driven by minimizing the wirelength. In principle, this is achieved by minimizing the number of nets cut ($\Rightarrow$ min-cut) by a partition. However, partitioning a circuit is a NP-hard problem [SH86]. Therefore, different heuristics were developed for this task [KL70, SK72, FM82, GB83, Kri84, Saa93, LLLC96, DD96b, DD96a, KAKS97, AHK97, CLL$^+$97, ACH$^+$97]. Beside the improvement in partitioning the circuit, the partitioning of the placement area was also improved. The first min-cut placers divided the placement area in two parts (bi-partitioning) in each step of the recursive placement process. [Bre77a, Bre77b, Cor79, Lau79, SH80, BH83, DK83, DK85, LD86, Zim88, SC88]. Later on, four parts [SK87, SK88, Apt90, HK97], and even eight parts [San89, Vij89, ML90] were used. Modern min-cut placers are for example Capo [RPA$^+$05], Dragon [TYC05], and Feng-Shui [AOL$^+$05].

2.1.4 Stochastic Placement

Stochastic placers combine the wirelength and the module overlap in one cost function, and minimize this cost function with stochastic methods. Stochastic methods means to create randomly sets of placements in a sequence of iterations. In the end, the placement with the lowest cost function is chosen as the result. Stochastic placers can easily extend the cost function in order to consider different objectives or various constraints. Moreover, stochastic placers are able to escape from local minima, and are even able to find the optimal solution for the placement problem. However, stochastic optimization in general needs a lot of samples (placements), and thus, stochastic placers are only practicable for circuits with a low number of modules. In principle, there are two main methods of stochastic optimization: simulated annealing and evolutionary algorithms.

Simulated Annealing

Simulated Annealing [KGV83] follows the annealing process in metallurgy: a hot metal is cooled (over time) such that in the end, it is most perfect (one crystal, no defects). As an optimization method, Simulated Annealing starts with an arbitrary start configuration (placement). Over the iterations, new configurations are created randomly by so called “moves”. A move for a placement can be to choose randomly a module, and to change randomly its location. Each new configuration is given a cost, and a decision is made if the new configuration is accepted, and thus replaces the best-so-far configuration. This decision is done based on the cost of both configurations, and based on the current temperature. The temperature is high at the start, and is decreasing over the iterations. As a result, worse configurations are accepted at the start of the optimization process, in order to escape from local minima. At the end, only better configurations are accepted. The method of decreasing the temperature affects highly the quality of the solution [Whi84, HRSV86, LD88, BKT93].

The authors of [RSV85, vLA87, Sec88, OvG89, AK89] showed that simulated annealing is able to find the global optimum. Moreover, the basic operations of the optimization techniques are easy to implement. Hence, this technique was very popular for placement in the past [SSV85, NSS85, SSV86, WL86, Sec88, WLL88, MFNK96, NFMK96]. However, the number of configurations necessary to find the optimum increases dramatically with the complexity, i.e., the number of modules per circuit. Therefore, different heuristics were used along with simulated annealing to cope with the increasing number of modules per circuit [MG88, HCC92, SKK$^+$93, SS95, SW97]. A typical representative of a stochastic placer is Timberwolf [SS93]. Today, simulated annealing is rarely used to place circuits with millions of modules.

Evolutionary Algorithms

Evolutionary algorithms use mechanisms inspired by biological evolution: heredity, mutation, selection, and survival of the fittest. In placement, evolutionary algorithms start by creating a set of random placements. In an iterative process, new placements are created based on current placements (heredity), and based on random changes (mutation). Then, the new placement are selected according to their cost. Over the iterations, the better placements survive, and at the end, a good placement is found. In principle, the basic operations of evolutionary algorithms are simple, and the optimization can be run in parallel using numbers of computers. However, the runtime is still high for modern circuits. Evolutionary algorithms for placement are presented in [CP86, CP87, KB89, SM90, KB91, RR96, EK97].

2.1.5 Analytical Placement

Analytical placers are based on an analytical cost function, which is continuous and in most cases differentiable. The minimum of the analytical cost function is determined by numerical optimization. Mostly, the cost function represents the wirelength, and sometimes it is a combination of wirelength and overlap. Depending on the cost function, analytical placers can be subdivided in linear, quadratic and non-linear placers.

Linear Placement

Linear placers are using a linear cost function, and remove the module overlap by linear constraints between the modules. This gives a linear program. However, such programs have a high computational complexity. Hence, linear placers like [WM87, HWM86, WM88, JK89, RC06] can only be used for circuits with a low number of modules. The analytical cost function in linear placement can be non differential (e.g., using the absolute value function). In all other analytical placement approaches, the cost function is differentiable.

Quadratic Placement

All quadratic placers represent the wirelength in a quadratic cost function Γ:

$$\Gamma = \frac{1}{2} \sum_{i,j} w_{x,ij}(x_i - x_j)^2 + w_{y,ij}(y_i - y_j)^2 \tag{2.1}$$

$\mathbf{p_i} = (x_i, y_i)$ is the position of module i. Γ is the sum of the weighted quadratic Euclidean distances between pairs of modules (i and j). The pairwise connections are called two-pin connections. To represent the wirelength by two-pin connections in Γ, a net model is necessary in quadratic placement. Next Section 2.3 gives an overview on net models in general, and on state-of-the-art net models for quadratic placement. Amongst others, this thesis presents a novel net model for quadratic placement.

Representing the positions of all N movable modules in vector $\mathbf{p} = (x_1, x_2, ..., x_N, y_1, y_2, ...y_N)^T$, the sum notation of the quadratic cost function (2.1) can be represented in a matrix-vector notation:

$$\Gamma = \frac{1}{2}\mathbf{p}^T\mathbf{C}\mathbf{p} + \mathbf{p}^T\mathbf{d} + const \tag{2.2}$$

Matrix $\mathbf{C}$ represents the connections between movable modules, and vector $\mathbf{d}$ reflects the connections between movable and fixed modules. Fixed modules are for example I/O pins (input/output pins). By minimizing Γ, quadratic placers obtain the module positions $\mathbf{p}$ with minimal netlength, which is the optimal placement. Since minimizing just the netlength results in a lot of module overlap, quadratic placers need a method to reduce the overlap. Depending on this method, quadratic placers can be subdivided into three categories: based on eigenvalues, based on partitioning, and based on forces.

Eigenvalue-Based Quadratic Placement

Quadratic placers based on Eigenvalues assume that all modules are movable, i.e., $\mathbf{d} = \mathbf{0}$ in (2.2). To reduce the module overlap, and to spread the modules on the placement area, these placers are using the constraint $\mathbf{p}^T\mathbf{p} =$ *const*. Combining this constraint with the quadratic cost function Γ by Lagrangian relaxation gives a new function, whose minimum is found by setting its derivative (with respect to x_i and y_i) to zero. This results in $\mathbf{C}\mathbf{p} - \lambda\mathbf{p} = \mathbf{0}$, which is similar to determining the Eigenvalues and Eigenvectors of $\mathbf{C}$. Then, the module positions are given by the Eigenvectors with the lowest Eigenvalues. Eigenvalue quadratic placers are for example [Hal70, Ott82a, Ott82b, FYSK83, Bla85a, Bla85b, FK86]. Since computing Eigenvalues and Eigenvectors is complex, quadratic placers based on this technique are rarely used to place state-of-the-art circuits with millions of modules.

Partitioning-Based Quadratic Placement

In order to reduce the module overlap, partitioning-based quadratic placers divide recursively the circuit and the placement area, and assign parts of the circuit to parts of the placement area. In contrast to min-cut placers, which use a similar technique for placement, partitioning-based quadratic placers minimize a quadratic cost function in each step of the recursive placement process. In quadratic placement based on partitioning, different techniques are used to partition the placement area, to partition the circuit, and to hold the modules in the placement region to which they are assigned.

The authors of [WWM82, Wip85] presented a placer, which first places the modules by minimizing the quadratic cost function, and then assigns modules to placement regions using a technique similar to min-cut. In [CK83, CK84], a method is described, which recursively partitions the placement area in two regions. In each iteration of recursion, the positions of the modules are used to partition the circuit, and to assign the modules to placement regions. To place the modules in one region, the modules of the other regions are fixed, and linear constraints (center-of-mass constraints) are used to spread the modules. PROUD [TKH88a, TKH88c, TKH88b] is similar to this technique, but does not utilize linear constraints. To spread the modules in one region, the fixed modules of the other regions are projected to the border of the current region. With the recursion, the placement regions, and the number of modules assigned to them are continuously decreasing. By placing only the modules in one region, and fixing all other modules, the placement problem is solved more and more locally. This will decrease the quality of the solution. In contrast to this, Gordian [KSJ88, KSJ89, Kle89, KSJA91] places all modules concurrently in all iterations of the recursive partitioning process. The partitioning is driven by the module positions. To hold the modules, which are assigned to one placement region, in this region, Gordian uses center-of-mass constraints. GordianL [SDJ91, Sig92] improves the method for partitioning the placement area, and introduces weights in the quadratic cost function, which are used for linearization the quadratic wirelength.

BonnPlace [Vyg97, BS05], and hATP [NRA$^+$06] partition the placement area in four regions in each step of recursive placement process. A min-cost-max-flow is used to partition the circuit, and to assign modules to placement regions. To hold the modules in their placement regions, BonnPlace and hATP use center-of-mass constraints, and so called “terminals”. These terminals arise while cutting the nets by partitioning. In other words, the terminals connect two nets of two partitions, which where formerly one net in one partition. The terminals are located at the border between two partitions, are treated as fixed modules, and results in that the modules in each placement partition stay within its partition. In addition, with the fixed terminals, each placement partition can be placed concurrently using different CPUs. This improves runtime, but advanced methods for positioning the terminals are necessary in order to prevent a decline in the placement quality.

In general, partitioning quadratic placers are able to place modern circuits in reasonable runtime. Since they reduce the module overlap by partitioning, and mostly ignore the module dimension here, they are problematic if the modules are of different dimension like in mixed-size circuits.

Force-Directed Quadratic Placement

The two-pin connections used in (2.1) for the quadratic cost function Γ can be viewed as elastic springs. This creates a spring system, and Γ represents the total energy of the spring system. The derivative of Γ is the “net” force, created by the springs: $\mathbf{F_{net}} = \mathbf{Cp} + \mathbf{d}$. Setting this force to zero gives the module positions with minimal wirelength, which equals the equilibrium state of the spring system. In other words, the springs, i.e., the two-pin connections, of quadratic placement create a force, which attracts the modules. Force-directed quadratic placers utilize an additional force $\mathbf{F_{add}}$ to spread the modules on the placement area. This spreading is done in a sequence of placement iterations. Each iteration starts with a given placement. Then, an additional force is determined. Setting the sum of the net force and the additional force to zero results in a system of linear equations. This system can be solved efficiently with respect to $\mathbf{p}$. At the end of each placement iteration, the modules are placed to the positions described by $\mathbf{p}$.

Different approaches exist for the additional forces. In [FCW67], the additional force is modeled in that all modules are repelling each other. However, this results in a high number of additional forces. To reduce the computational complexity, other approaches utilize repelling forces only between unconnected modules. In [Sca71, Qui75, QB79, AJK82, JJA83, Kir84, For87, Jus87], the repelling force is constant over the distance between the not connected modules. In [FCW67, QB79, Kir82, Waw88], the repelling force is reciprocal to the distance. Another modification is to model the overlaps between the modules rather than the modules themselves as the source for the repelling force. In [Sca71, Shu75, Rob83, SD85, SB87, AA88, KKM91] overlaps between modules are repelling each other. The overlap between modules and the border of the placement region is modeled in [FCW67, Shu75, KKM91] as the source for the repelling force. In [Joh87], the triangulation of the placement area based on the module positions is used to determine a force, which spreads the modules on the placement area.

Modern force-directed quadratic placers like Eisenmann's approach [EJ98, Eis99, Obe05], FDP [VKV04, VK05a, VK05b, KV06], FAR [HMS02b], mFAR [HMS05], FastPlace [VC05, VPC06, VPC07], and RQL [VNA$^+$07], have in common to use the distribution of the modules on the placement area to determine one additional force per module. This force drives the modules away from high density regions towards low density regions. The above mentioned modern force-directed placers differ in the way how the additional force is implemented, i.e., in the way how the force is determined and modeled. Since this thesis presents a force-directed placer, details and differences of modern force-directed placers are described in the following; Figure 2.2 gives an overview.

Placer	Controlling Force Hold Force	Spreading/Perturbing Force Move Force
Eisenmann's approach, FDP*	Const. Force, Potential	
FastPlace, RQL	Fixed Points, Bin Utilization	
FAR	Fixed Points	Const. Force, Potential
mFAR	Fixed Points	Fixed Points, Bin Utilization
Kraftwerk	**Const. Force**	**Target Points, Potential**

Figure 2.2: Implementation of the additional force in modern force-directed quadratic placers. *FDP uses two more forces, but they are not necessary to spread the modules on the chip. A dark gray box means that heuristics are necessary. A light gray box means low controllability.

Eisenmann's approach is based on the idea that modules are positively charged, the placement area is negatively charged. Thus, the modules repel each other, and the modules are attracted by the placement area. The distribution of the charges on the placement area is used to determine an electrostatic potential. For each module i, the gradient of the potential is determined, and the gradients are accumulated in the additional force over the placement iterations. The additional force in Eisenmann's approach is modeled as constant force, i.e., the force does not depend on $\mathbf{p}$.

Using a constant force is one way to model a force. Another way to model a force is to use fixed points (each located at $\mathring{p}_i$), and connect each module to its fixed point by an elastic spring having the strength s_i. This spring creates the force then.

$$\mathbf{F}_\mathbf{i}^\mathbf{spring} = s_i \left(\mathbf{p_i} - \mathring{\mathbf{p}}_\mathbf{i} \right) \tag{2.3}$$

The authors of [HMS02b] showed that using fixed points are a generalization of using a constant force, and they showed that fixed points control the placement better than constant forces do. In principle, the controllability is improved because each module is moved at most to its fixed point in each placement iteration. Using a constant force, the movement is not limited.

FDP is similar to Eisenmann's approach in that the gradients of the potential are accumulated in a constant force to spread the modules on the chip. In addition, FDP used two forces to stabilize the placement algorithm, and to improve the netlength. These two forces are modeled by fixed points in FDP. Similar to Eisenmann's approach, FAR utilizes an electrostatic potential to determine a force, which spreads the modules on the chip. This additional force is modeled as a constant force. Instead of accumulating the spreading force over the iterations, FAR uses a second additional force for each module to control the placement process. This force is modeled by fixed points and is determined by achieving force equilibrium at the start of each placement iteration. The main difference between FAR and mFAR is that mFAR uses a local bin utilization to determine the spreading force, and the spreading force is modeled by fixed points. Using a local bin utilization, the spreading force has a local view, as the force of one module depends only on the surrounding modules. In contrast to this, the (spreading) force in Eisenmanns' approach, FAR, and FDP has a global view, i.e., the force of one module depends on all modules. This is because the force is based on potential formulation there, and the potential represents all modules.

Instead of accumulating one additional force over the placement iterations, or using two additional forces, FastPlace and RQL are using a different method to spread the modules. In each placement iteration, a local bin utilization is determined similar to mFAR. The additional force for one module i is then determined as follows. Module i is temporary placed to the position determined by the local bin utilization. This can be viewed as a local diffusion process. Then, the force is determined, which holds module i at its temporary position. After that, module i is put back to its original position. After determining the additional force for all modules, the new positions for all modules are obtained by setting the sum of the net force and the additional force to zero. The additional force is modeled by fixed points. In FastPlace, the fixed points are located at the border of the placement regions. RQL uses a location between the border and the module position. In addition, RQL modulates the additional force, which means that for some modules, the additional force is ignored. With this, the modules are reordered during placement, which can improve the netlength. On the other hand, the convergence of the placement algorithm can be harmed.

In summary, fixed points are widely used in modern force-directed quadratic placers. The locations of the fixed points are all determined in that a force is given. This force is to be represented by the spring connection between each module and its fixed point. In this case, where the force is given, a good heuristic is necessary to obtain suitable locations of the fixed points. This is a well-known critical problem of using fixed points [HMS02b, HMS05, VNA+07].

$$\mathbf{F_i^{spring}}\Big|_{\mathbf{p_i}=\mathbf{p_i'}} = \mathbf{e_i} \quad \Leftrightarrow \quad \mathring{\mathbf{p}}_\mathbf{i} = \mathbf{p_i'} - \frac{1}{s_i}\mathbf{e_i} \tag{2.4}$$

In (2.4), the force $\mathbf{e_i}$ of module i is given, and the module is located at $\mathbf{p_i'}$, i.e., $\mathbf{p_i} = \mathbf{p_i'}$. If the strength s_i of the spring is too low, the fixed point $\mathring{\mathbf{p}}_\mathbf{i}$ is too far away from the module position $\mathbf{p_i'}$, and the force is modeled like a constant force, resulting in low controllability. If the strength s_i is too high, the fixed point is too near to the module, and the module movement is highly limited. Thus, all modern force-directed placers, using fixed points, rely on heuristics for good values of s_i. The force-directed quadratic placer Kraftwerk, as presented in this thesis, also uses fixed points (called “target points”), but does not depend on critical heuristics. Rather, the locations of the target points are directly given by the gradients of an electrostatic potential. In other words, not the force is given, but the location of the target points. In Kraftwerk, two forces are used: a moving force, modeled by target points, and a hold force, modeled as a constant force. The constant hold force does not reduce controllability of the placement process, but enforces the convergence.

Nonlinear Placement

Nonlinear placers are based on a nonlinear cost function, which is even not quadratic. Placers based on nonlinear cost functions have appeared in the recent years, after developing an efficient representation of the wirelength by a log-sum-exp function [NDS01]. The major drawback of nonlinear placers is that nonlinear numerical optimization takes high runtimes. Nonlinear placers differ mainly in the way how the module overlap is removed.

Density-Driven Nonlinear Placement
Density-driven nonlinear placers are using the distribution of the modules on the placement area (i.e., the module density at various points) to determine a nonlinear function, which represents the module overlap, and which is continuous and differentiable. This function is combined with the wirelength function in a total cost function, and the total cost function is minimized by nonlinear numerical optimization. In this way, the modules are iteratively spread over the placement area. Examples for density-driven nonlinear placers are APlace [KW05a, KRW05], mPL [CCS05], and NTUPlace [CJH+06].

Nonlinear Placement Based on Pseudo Nets
Nonlinear placers based on pseudo nets are using additional “pseudo” nets (one for each module). This is similar to the fixed point approach used in force-directed quadratic placement. Minimizing the wirelength of the nets and the pseudo nets, the modules are spread iteratively over the placement area. In each placement iteration, Vaastu [AM07] is using a min-cost-max-flow to assign modules to placement regions. Then, the pseudo nets are created between each module and the center position of the placement region to which the module is assigned. In other words, and considering force-directed quadratic placement, the fixed points of the

pseudo nets are determined by a min-cost-max-flow approach in Vaastu. Other nonlinear placers using pseudo nets are not known up to now.

2.1.6 Warping Placement

Placers based on warping start with an initial placement, and are using approaches of computational geometry to deform the placement area, and thus moving the modules indirectly. The deformation of the placement area is driven by minimizing the wirelength and the module overlap. Placers based on warping are for example [XMFR04, XR07, CS07]. To obtain the initial placement, warping placers usually follow quadratic placement and minimize the quadratic wirelength.

2.2 Multilevel Approach

To place "large" circuits, i.e., circuits with a high number of modules, some placement approaches are following a hierarchical approach. Min-cut placers, placers based on cluster-growth, and some partitioning placers are per se hierarchical, because not all modules of the circuit are placed simultaneously in all placement iterations.

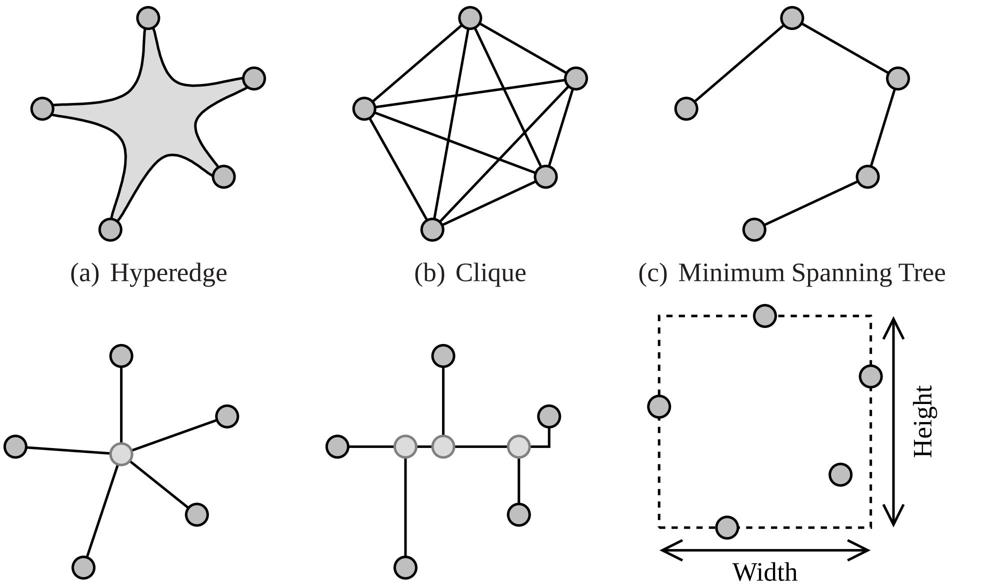
A general hierarchical approach to cope with "large" circuits is the multilevel approach, which can be used for all placement techniques. Starting from the "flat" circuit, which consists of all modules, the modules are clustered over a few levels during the coarsening phase. Then, the coarsest circuit is placed. In the refinement phase, the placement of the previous level is used as input, the clusters are declustered, and the new "refined" circuit is placed. The refinement is done until the flat circuit is placed. Since only some placement iterations are spent in each level of refinement, and in particular only some iterations for the flat circuit, the runtime decreases with the multilevel approach. However, the major drawback of the multilevel approach, and of all hierarchical approaches in general, is that a good heuristic is necessary to partition or cluster the circuit. This is because optimal partitioning is an NP-hard problem [SH86]. In addition, using a hierarchical approach, the placement problem is solved more locally then using a flat approach, where all modules are placed concurrently in all placement iterations.

2.3 Net Models

The previous sections described different techniques to solve the placement problem. The general objective of the placement problem is to minimize the total length of all nets. This objective is used because a placement with minimal netlength is usually optimal also in other objectives like area consumption, routability, timing (length of the critical path), etc. This section describes how to measure the length of one net. There, the net is represented by graphs, different net metrics are shown, and net models necessary for quadratic placement are presented.

2.3.1 Graphs and Metrics

In principle, one arbitrary net consists of N pins, and each pin $i = 1, 2, ..., N$ is located at (x_i, y_i). The property of a net is that all its pins must have the same electric potential. Consequently, all pins of one net must be connected by a wire. Using graph theory, the pins are nodes, and the connections between the nodes are represented by edges (each connecting two nodes), or by a hyperedge (each connecting two or more nodes).



(a) Hyperedge (b) Clique (c) Minimum Spanning Tree

(d) Star (e) Steiner Tree (f) Half perimeter wirelength (HPWL)

Figure 2.3: Different net models.

Figure 2.3 shows different net models. The hyperedge net model, as displayed in Figure 2.3(a), consists of one hyperedge, connecting all pins of the net. All other net models are using two-pin connections to represent the net. There, each two-pin connection, i.e., each edge $e = (i, j)$ between two pins i and j, is associated a cost, and the cost represents the distance between both pins. Using the Manhattan norm, which is based on just using horizontal and vertical wires, the distance between both pins is $|x_i - x_j| + |y_i - y_j|$. In the quadratic Euclidean norm, the distance is $(x_i - x_j)^2 + (y_i - y_j)^2$. This quadratic norm is used in the next section addressing net models for quadratic placement.

The clique net model (see Figure 2.3(b)) uses all possible two-pin connections of one net. The number of two-pin connections is $0.5 \cdot N \cdot (N - 1)$. The minimum spanning tree model [Pri57], which is displayed in Figure 2.3(c), is driven by using a minimal set of edges, whose total cost is minimal. Here, there are $N - 1$ number of edges. However, the construction of the minimum spanning tree needs some runtime, and the runtime complexity is more than $O(N)$ [Eis97]. The star net model (see Figure 2.3(d)) uses one additional star pin, which is located in the center of the net, and connects each pin with the star pin. This results in N edges, and the runtime complexity is $O(N)$. The Steiner tree net model, as shown in Figure 2.3(e), uses several additional pins, and is driven by connecting all pins by horizontal or vertical edges only. In the minimal Steiner tree, the edges are chosen such that the total cost of all edges is minimal. Finding such an optimal Steiner tree is known to be a NP-hard problem [GJ77]. However, there exist numbers of algorithms, which find a near-optimal Steiner tree in practicable runtime [Han66, Hwa79, Ser81, CRS88, HVW90, GRSZ94, Chu04]. Since routing of a net is similar to constructing the minimal Steiner tree, the routed wirelength, i.e., the wirelength after routing, is best approximated by length of the minimal Steiner tree. However, routing is more complex than just constructing the minimal Steiner tree, as more things have to taken into account in routing. For example, there is only a limited number of routing tracks available in a chip, which limits the resources for routing. Or not only the wirelength is to be minimized in routing, but also the number of vias.

The half-perimeter wirelength (HPWL), as illustrated in Figure 2.3(f), is rather a metric for the netlength, than a net model. Here, “half-perimeter” means the half-perimeter of the smallest rectangle enclosing all pins of the net. The width of this rectangle is given by $w = \max x_i - \min x_i$, and the height is given by $h = \max y_i - \min y_i$. Then, the HPWL is $w + h$. The HPWL equals the length of the minimal Steiner tree for nets with two or three pins [Han66]. For nets with four and more pins, the HPWL is a lower bound. Since most of the nets of a circuit are two and three pin nets, the HPWL is an efficient estimation of the length of the minimal Steiner tree [Chu04], and consequently, it is an efficient estimation of the routed wirelength [Ser81, SKAS88]. Here, efficient means that the HPWL offers low runtime and good approximation.

2.3.2 Net Models for Quadratic Placement

Quadratic placement is based on two-pin connections, and minimizing a quadratic cost function (2.1), which represents the sum of the quadratic lengths of the two-pin connections. Since the runtime complexity of determining suitable two-pin connections is practicable in the clique and the star net model, these net models are used widely in quadratic placement. Traditionally, the weights of the two-pin connections are used to linearize the quadratic length, and to approximate the quadratic cost function to the HPWL metric.

Considering one net with N pins, a weight of $1/N$ in the clique net model adapts its quadratic costs to the cost of the corresponding star net model [Sig92, VC05]. Hence, clique and star net model can be used interchangeably. The authors of [Vyg97, BS05] use an additional weight of $1/N - 1$ for each net, in order to prevent that nets with a high number of pins are dominating the quadratic cost function. In [SDJ91, Sig92], the additional weight for each net is $2/N$, and a linearization weight for each two-pin connection is used, in order to adapt the quadratic cost to the HPWL.

Since the clique and the star net models have different characteristics, and both can be used concurrently, there is a trade-off between both net models [EJ98, Eis99, VC05]. The clique net model has no additional star pin, but a complexity of $O(N^2)$ in the number of two-pin connections. The star net model introduces one additional star pin per net, but has only $O(N)$ two-pin connections. To minimize the quadratic cost function in short runtime, the number of two-pin connections, and the number of pins should be as low as possible. In an average circuit, most of the nets have two or three pins, and nets with a high number of pins are rare. Hence, the clique model is used for small nets, i.e., for nets with a about six or less pins, as the number of two-pin connections is reasonable here. For big nets, the star net model is used, as the number of two-pin connection is low here, and the number of additional star pins is reasonable. Using clique and star net models concurrently in a circuit gives the hybrid clique/star net model.

The authors of [BS05] propose a net model suitable for partitioning quadratic placers, which is based on the star net model, but introduces additional pins (so called "terminals") for those nets, which cross the border of two placement partitions. In [OJ04a, Obe05], a method is described, which integrates the minimal Steiner tree in the quadratic cost function. This is used to obtain better timing-driven placements. However, determining a minimal Steiner tree is time consuming.

This thesis presents a new net model, which accurately represents the HPWL in the quadratic cost function. Compared to a hybrid clique/star net model, the new net model offers better placements in lower runtime.

2.4 Routability-Driven Placement

In the layout synthesis of an integrated circuit, the modules are placed first, and the nets are routed then. These are two separate steps, mostly done by two different computer programs. Placement traditionally targets to minimize the total wirelength, which in general improves routability. However, the placed circuit may not be routable, because there are so called "congested regions" on the chip, where too many wires are necessary to route the nets than routing tracks are available. In other words, the routing demand, created by the nets, exceeds the routing supply, given by the routing layers. Due to such congested regions, the circuit has a high routed wirelength, or is even not routable. Therefore, besides minimizing the total wirelength, placement has to be driven by routability, which means to remove the congestions during placement. To do routability-driven placement, two problems have to be solved. First, a fast and accurate method to estimate the congestions is necessary. This is because the exact informations about congested regions would be given after routing, but routing itself takes enormous runtime. Second, the congestion estimation has to integrated effectively in the placer. This thesis presents novel solutions for both problems. Therefore, the state-of-the art in congestion estimation and in the integration in placement is described next.

2.4.1 Congestion Estimation

Assuming a constant routing supply, congestion estimation means to estimate the routing demand. Most published methods to estimate the routing demand are using a grid structure to divide the chip area into a number of bins, and estimate the routing demand in each bin.

Based on the bounding box of one net, i.e., the smallest rectangle enclosing all pins of one net, the authors of [lEC94] presented a simple method to estimate the routing demand in one bin: the routing demand of one net in one bin depends on the overlap between the bounding box of the net and the bin. Another simple technique to estimate the routing demand in one bin is to use the pin density within this bin [BR02, ZD02]. A widely applied technique to estimate the routing demand is to use a routing model, which models possible routes of each net. The number of possible routes crossing the border of a bin reflects the routing demand in the bin. In most approaches based on routing models, multi-pin nets are broken down into two-pin connections by using a minimum spanning tree. Then, for each two-pin connection, different routes with different number of bends are modeled. The authors of [LKS02] use all possible routes for each two-pin connection. This probabilistic routing model is improved in [KX03, SZJ06] by adjusting its result to the result obtained by routing. The authors of [WBG04] state that one- and two-bend routes are enough to model the routing demand. In [PC06], a fast global router is proposed, which uses different Steiner Trees to model the possible routes of each net. In [YKS01, YKS02, HMS02a], the maximal routing demand of a circuit is estimated based on Rent's Rule [LR71]. Another technique to estimate the routing demand is the analysis of the distribution of the number of nets per bin [WYES00].

2.4.2 Integration in Placement

Estimating the routing demand in an efficient way is the first step to optimize routability during placement. The second step is to integrate the estimation of the routing demand in the placement algorithm, in order to remove the congestions and to improve routability. Since the congested regions are characterized that the routing demand of the nets is higher than the supply by the routing layers, there exist two main approaches to optimize routability. The direct approach reduces the routing demand in congested regions, and the indirect approach increases the routing supply in congested regions. The routing supply can be increased, because modules block some routing layers, and with a lower module density, more free space is available in the routing layers. The routing demand can be decreased by replacing modules, such that the nets connected to the modules are moved out of the congested regions. The direct approach is often used as a post-process to tune an already placed circuit for routability. A post-process utilizing Simulated Annealing is described in [lEC94, HMS02a, WS99]. A flow-based method is presented in [WYS00, WS00]. Linear programming is used in [LWH03].

The indirect approach to optimize routability is mostly used during placement. In [HYH$^+$01, BR02], a quadratic placer is described, which inflates modules in congested regions. The authors of [PBS98] present a quadratic placer, which reduces module density in congested regions by growing these regions. In [YCS03], a min-cut placer is shown, which allocates white space, i.e., reduces module density, in congested regions during top-down placement.

In the following, routability optimization in state-of-the-art placers is described. mPL [LXK$^+$04, LXK$^+$07] is a multilevel analytical placer based on non-linear optimization. mPL estimates the routing demand based on a two-pin connection routing model developed in [CCPY02]. Routability is optimized in global placement by moving certain modules out of congested regions in order to reduce the routing demand there. In final placement, a white space allocation (WSA) method is used, which is based on recursively partitioning the placement area, and shifting the cut lines according to the routing demand. Thus, mPL utilizes the direct approach during global placement, and the indirect approach after wards in detailed placement.

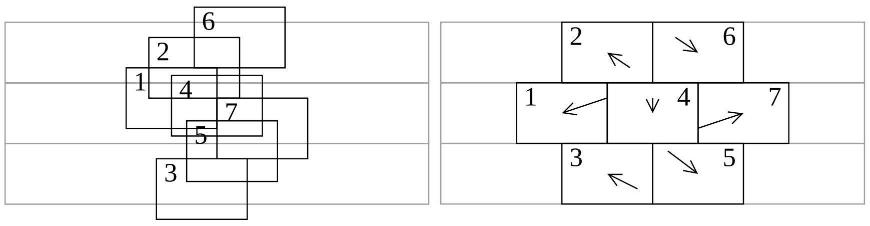
ROOSTER [RLM06], as a feature of Capo 10, is a min-cut placer. The placer models nets by Steiner trees [Chu04], and estimates the routing demand by a probabilistic routing model [WBG04]. The cut lines are shifted during global placement based on the routing demand. During final placement, the WSA method of [LXK+04] is used. Therefore, ROOSTER applies the indirect approach to optimize routability.

APlace [KW05b] is a multilevel analytical placer based on non-linear optimization. APlace estimates the routing demand by a probabilistic routing model [KX03]. Routability is optimized during global placement by decreasing module density in congested areas, i.e., by the indirect approach.

2.5 Final Placement

The global placement approaches proposed in Section 2.1 spread the modules roughly on the chip, while considering different objectives like total wirelength and routability. After global placement, final placement is done. Final placement itself consists mostly of two consecutive steps: legalization and detailed placement. In legalization, the remaining overlap of the global placement is removed, and the modules are aligned to a row or grid structure if necessary. In detailed placement, the legal placement is improved such that the total wirelength is further reduced, or more complex objectives like design for manufacturing (DFM) [GKP05] or design for yield (DFY) [ABD+07] are considered. The common approach in detailed placement is to use small sliding windows in order to capture a low number of modules (about 10 modules), and to do different transformations on this set of modules. For example, single modules are rotated, pairs of modules are exchanged, or all modules in the set are permuted [CKM00, CX06, LXK+07, PVC05, RPA+07]. In [KTZ99, BV00], a detailed placement approach suitable for standard cell circuits is described. There, the modules in each row are placed such that their total HPWL netlength is minimized. The ordering of the modules is not changed here.

Since this thesis describes new approaches for legalization, this section focuses on the state-of-the-art techniques for legalizing a global placement. To preserve the global placement as far as possible, the common objective of legalization is to move the modules as little as possible. While most global placement approaches can deal with different circuit types like standard cell circuits, macro cell circuits, and mixed size circuits, legalization approaches differ in the circuit type for which they are applicable. This difference in legalization is because of the different “design rules” for each circuit type. So, the modules of FPGA circuits, and the modules of sea-of-gates circuit have to aligned to a grid structure. The modules of standard cell circuits have to be aligned to rows. And the modules of macro cell circuits have not to be aligned to rows. These design rules are mostly ignored during global placement as the modules are spread just roughly on the placement area. Because of the difference in the application of the legalization approaches, the modules of global placement are now called standard cells, or macros. In the following, state-of-the-art approaches for legalizing standard cell circuits are proposed. Most of the approaches are also applicable for FPGA circuits, and for sea-of-gates circuits. In addition, modern methods for legalizing macros are described. In Chapter 7, novel approaches for legalizing these two circuit types are presented.



(a) Global Placement (b) Legal Placement

Figure 2.4: Global and legal placement of standard cells.

2.5.1 Legalization of Standard Cell Circuits

Figure 2.4 displays a global and legal placement of a (very small) standard cell circuit. Various approaches exists for legalizing standard cell circuits. Domino [DJA94] is based on network flow, shreds cells into subcells and rows into places. Here, all subcells and all places have the same height and width. The subcells are placed, i.e., assigned, to places by solving a min-cost-max-flow. The authors of [BV04, BPV04] present a similar method as Domino, but assign sets of modules to row regions by a min-cost-max-flow. Fractional Cut [YKM$^+$03] is a two stage approach: first the cells are assigned to rows by dynamic programming, then the cells of each row are packed from left to right. The authors of [KMR04] also present a two stage approach: first the cells are assigned to the rows by heuristical cell juggling, then the cells of each row are placed by finding a shortest path in a graph. Mongrel [HL00] uses a greedy heuristic to move cells from overflowed bins to under capacity bins in a ripple fashion based on total wire length gain. Diffusion based placement migration is presented in [RPAV05] to remove cell overlap. In [LRAP07], computational geometry is used to spread the cells, and to align them to rows. NRG [SW97] uses simulated annealing for legalization.

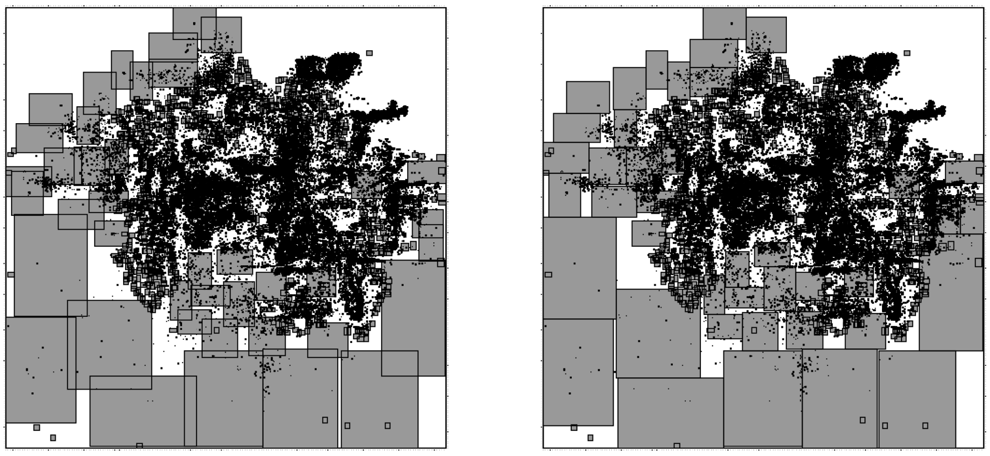
Tetris [Hil02] is a fast greedy heuristic, which is used widely [LXK$^+$07, KW05a, KLA$^+$04], for example. In [LK03] a similar approach to Tetris is described. Tetris sorts the cells first, and legalizes one cell at a time then. Legalizing one cell is done by moving the cell over the rows, and within the rows by moving the cell over free places. This movement is done until the nearest free place is found. Once a cell has been legalized, it will not be moved anymore. This results in a high total cell movement during legalization.

2.5.2 Legalization of Macros in Mixed-Size Circuits

In pure macro circuits, which consist only of macros, legalizing can be driven by minimizing the area consumption, rather than the macro movement. Such legalization of macro circuits can be done for example with shape-functions [Ott83, SS91], sequence-pairs [MFNK95, MFNK96], or B*-trees [CCWW00, WC04, cCYc$^+$07].

However, mixed-size circuits consist of a few macros, and millions of standard cells. Figure 2.5 displays a global and a legal placement of such a mixed-size circuit. To respect the standard cells, the macros of mixed-size circuit have to be legalized such that their total movement in minimized. In Figure 2.5(b), the macros are legalized in this way.

Different approaches exist for legalizing macros in mixed-size circuits. The authors of



(a) Global Placement (b) Legal Placement for Macros

Figure 2.5: Legalization of macros in mixed-size circuits. Gray rectangles represent macros, black clouds represent the standard cells.

[CCY03, VPC06] are using a low-temperature Simulated Annealing approach in combination with sequence-pairs. Although Tetris was introduced in the previous section as a legalization approach for standard cell circuits, it can also be used for legalizing macros [KLA^{+}04, CX06]. A direct approach to minimize the movement of the macros during legalization is to use linear programming (LP) [Vyg97, CX06, RC06]. Here, the objective is the total movement, and linear constraints between all (or almost all) pairs of macros assure that the macros do not overlap. In detail, two macros are not overlapping, if the distance between the center positions of both macros is large enough, either in x-direction, or in y-direction. Consequently, one constraint per macro pair in the LP is enough to assure that both macros do not overlap. However, the direction (x or y) of the constraint influences the objective of minimal movement. Different approaches exist to optimize the direction of the constraints. The authors of [Vyg97] utilize a branch-and-bound optimization approach. In [CX06], the initial directions of the constraints are determined based on the global placement. Then, a min-cut like technique is used to change some constraints from x- to y-direction, or vice versa.

Chapter 3

This Thesis

This thesis presents novel approaches for quadratic placement, both for global placement and for legalization[1]. All these approaches are driven by minimizing a quadratic cost function, which results in low runtime. In global placement, the total wirelength is minimized, while in legalization the total movement is minimized. In the following, different enhancements of the new quadratic placement approaches are summarized.

3.1 “Kraftwerk”: Force-Directed Quadratic Placement

The force-directed quadratic (global) placer “Kraftwerk”, as presented in this thesis, is characterized by the following enhancements over other force-directed quadratic placement approaches:

- The placement is represented in a general demand-and-supply system. Therefore, different circuit types are supported, e.g., standard cell circuits, macro cell circuits, mixed-size circuits, and circuits with fixed modules. In addition, the demand-and-supply system is used to optimize the routability of a placement.

- The additional force is separated into a hold force and a move force. This is new compared to Eisenmann’s approach, FDP, FastPlace, and RQL, but somewhat similar to FAR and mFAR.

- Both additional forces are implemented in a novel and systematic way. The move force is modeled by target points, and the locations of the target points are directly determined by the gradient of the potential of the demand-and-supply system. The hold force is modeled as a constant force, and decouples each placement iteration from its preceding iteration.

- Compared to other placement approaches, no heuristics are necessary in Kraftwerk to determine the locations of the target points. In addition, the target points enforce the control of the module movement. Since the potential represents all modules, and the

[1]Some content of this thesis is pre-published in [SJ06, SJ07a, SJ07b, SSJ08a, SSJ08b].

potential gives the target points of the move force, the move force has a global view. This means that the move force of one module depends on all modules. Furthermore, the constant hold force does not reduce controllability, but enforces convergence.

- As a result of the systematic force implementation, Kraftwerk converges such that the demand is adapted further to the supply in each placement iteration. This in principle means that the module overlap is reduced in each iteration. The consequence of the convergence is a fast, robust, and stable placement algorithm. In this thesis, the convergence is analyzed in theory and demonstrated by experimental results. In addition, the stability is shown by experimental results.
- A flat placement approach is followed, which means that the complete circuit is considered in each placement iteration. Compared to a multilevel approach, no heuristic for partitioning or clustering the circuit is necessary in the flat placement approach, and the solution space is not narrowed.

3.2 “Bound2Bound” Net Model

Besides a force-directed quadratic placer, this thesis also presents the new “Bound2Bound” net model, which can be used universally in all quadratic placers. The advantages of the Bound2Bound net model are:

- Exact representation of the half-perimeter wire length (HPWL) in the quadratic cost function. Based on experimental result in routability-driven benchmark suites, the HPWL is an efficient metric for the routed wire length.
- Compared to the clique net model, the number of two-pin connections is lower.
- Compared to the star net model, no additional star pins are introduced.
- Based on experimental results, the Bound2Bound net model offers lower runtime and better netlength than a hybrid clique/star net model.

3.3 Routability-Driven Placement

An important objective for global placement is to optimize routability. For this, two problems have to be solved. First, an efficient estimation of the congestions based on routing demand is necessary. Second, an effective integration of the congestion estimation in the placer is needed. Solutions for both problems are presented in this thesis.

3.3.1 “RUDY”: Routing Demand Estimation

The advantages of the routing demand estimation called “RUDY” is as follows:

- No grid structure is necessary, which means the placement area is not divided into bins.

- No routing model is used, which means the estimation is independent of the router.
- The estimation is accurate.
- The runtime is low.

3.3.2 Integration in Placement

The enhancements of the presented integration of RUDY in Kraftwerk are:

- Straight-forward integration by extending the demand-and-supply system of Kraftwerk.
- Concurrent reduction of the routing demand and increment of the routing supply in congested regions.
- One parameter models the characteristics of the router.

3.4 “Abacus” and “Puzzle”: Legalization

In addition to novel global placement techniques, including a net model and routability optimization, this thesis also addresses new approaches for legalizing standard cell circuits, and for legalizing macros in mixed-size circuits. The enhancements over other legalization approaches are as follows:

- The total quadratic movement is minimized. Other approaches are targeting the linear movement. Using the quadratic norm, the placement with minimal movement is found in low runtime.
- The relative order of the macros/standard cells is preserved. This means that considering two macros/standard cells a and b, with a left of b in the legal placement, then a was left of b in the global placement.
- “Abacus” determines the legal placement of standard cells by using efficient dynamic programming.
- “Puzzle” determines the legal placement of macros by quadratic programming. In addition, Tabu Search approach is used to determine if two macros are made overlap-free in x-direction, or in y-direction.

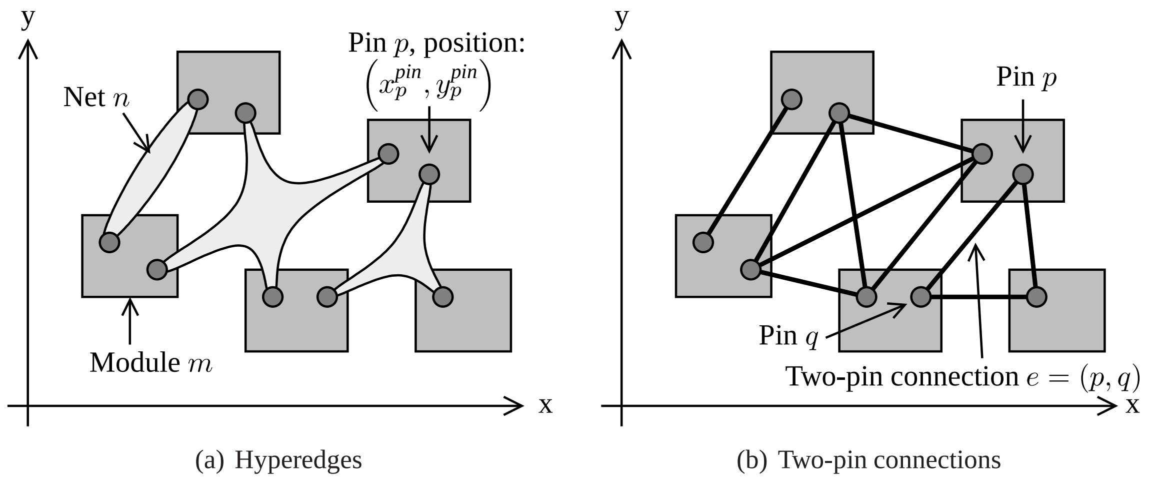


Figure 4.1: Circuit with hyperedges (a) and two-pin connections (b).

Chapter 4

Bound2Bound Net Model

Placement in general is based on the gate-level description of the circuit. This means, the circuit consists of modules (set $\mathcal{M}$), the modules have pins (set $\mathcal{P}$), and the pins are connected by nets (set $\mathcal{N}$). Each pin $p \in \mathcal{P}$ is located at $\left(x_p^{pin}, y_p^{pin}\right)$. Representing each net by one hyperedge gives the circuit as shown in Figure 4.1(a). In quadratic global placement, the nets are modeled by two-pin connections. This modeling is done by a net model, and results in that each net $n \in \mathcal{N}$ is represented by a set $\mathcal{E}_n$ of two-pin connections, as displayed in Figure 4.1(b). One two-pin connection $e = (p, q)$ connects pin p and q. The sum of the weighted quadratic Euclidean lengths of all two-pin connections gives the quadratic cost function Γ:

$$\begin{aligned}\Gamma &= \frac{1}{2}\sum_{n\in\mathcal{N}}\sum_{e=(p,q)\in\mathcal{E}_n} w_{x,p\,q}(x_p^{pin} - x_q^{pin})^2 + w_{y,p\,q}(y_p^{pin} - y_q^{pin})^2 \qquad (4.1)\\ &= \sum_{n\in\mathcal{N}} \Gamma_{n,x} + \Gamma_{n,y} \qquad (4.2)\end{aligned}$$

This cost function Γ can be separated in x and y-direction and in single nets, i.e., the cost $\Gamma_{n,x}$ is the cost of net n in x-direction. In the following, the focus is on $\Gamma_{n,x}$.

4.1 Clique/Star Net Model

Traditionally, the clique net model, or the star net model is used in quadratic placement. The clique net model utilizes all possible two-pin connections of a net. The star net model introduces an additional star pin per net, and connects each pin of the net to the star pin. With P pins in net n, the clique is equivalent to the star in the quadratic cost, if the clique cost is scaled with $1/P$ [LO73, Sig92, VC05]. Due to this equivalence of both net models, the focus is on the clique net model in the following. The quadratic cost of the clique net is:

$$\Gamma_{n,x} = \frac{1}{2}\sum_{p=1}^{P}\sum_{q=p+1}^{P} w_{x,pq}(x_p^{pin} - x_q^{pin})^2 \tag{4.3}$$

Different approaches exist for the connection weight $w_{x,pq}$. GordianL [SDJ91, Sig92] uses the following technique:

$$w_{x,pq}^{GordianL} = \frac{1}{P}\frac{2}{P}\frac{4}{|x_p^{pin} - x_q^{pin}|} \tag{4.4}$$

The first factor $1/P$ adapts the clique model to the star model. The second factor $2/P$ adjusts the number of connections of the clique to the number of connections in the corresponding spanning tree. With the factor $1/|x_p - x_q|$, the quadratic distance between both pins p and q is linearized.

The (quadratic) clique length (4.3) is just one metric for the netlength. The ideal metric for the netlength would be the routed wire length, as determined after final routing. However, placement is done iteratively, and in each iteration, the circuit would have to be final routed, which would take enormous CPU time. Experiments for routability-driven placement (see Section 6) reveal that the half-perimeter wire length (HPWL) is a very efficient metric for the netlength. The HPWL Γ_n^{HPWL} of the net n is defined by the width w_n and height h_n of the smallest rectangle, which encloses all $p = 1, ..., P$ pins of the net:

$$w_n = \max(x_p^{pin}) - \min(x_p^{pin}) \quad h_n = \max(y_p^{pin}) - \min(y_p^{pin}) \tag{4.5}$$

$$\Gamma_n^{HPWL} = w_n + h_n \tag{4.6}$$

Using GordianL's connection weight (4.4), the approximation error between the quadratic clique length $\Gamma_{n,x}$ and $\Gamma_{n,x}^{HPWL}$ is displayed in Figure 4.2. For two-pin nets, GordianL's approach results in no approximation error. This is due to the factor 4 in the last enumerator in (4.4). However, with increasing pins per net, the approximation error increases. On average, the approximation error is about 30%, and is too high to reflect the HPWL precisely in the quadratic cost function Γ.

An unpublished approach of Eisenmann uses the following two-pin connection weight:

$$w_{x,pq}^{Eisenmann} = \frac{1}{P}\frac{2}{P}\frac{10}{10 + w_n} \tag{4.7}$$

Figure 4.2 shows that the average approximation error of this approach also depends on the pins per net, and is increasing with the number of pins per net. In addition, Eisenmann's approach has a higher approximation error than GordianL's approach.

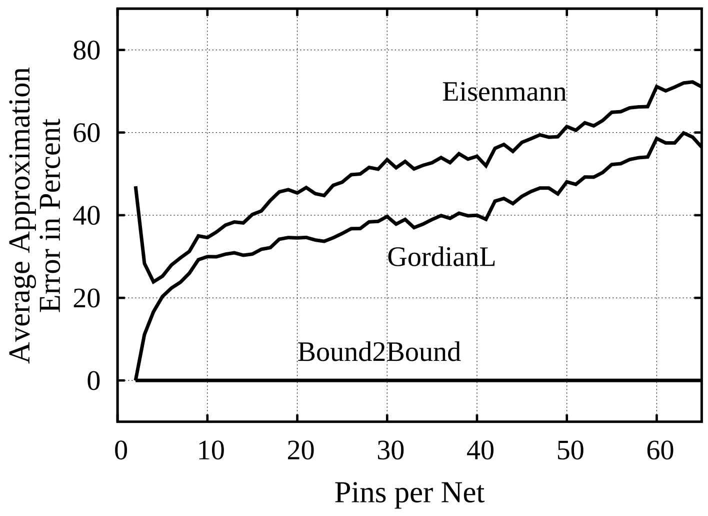


Figure 4.2: Approximation error between the quadratic cost function and HPWL, depending on the number of pins per net, and using different approaches for the connection weight $w_{x,pq}$. The statistic is based on 5.6 million nets of the ISPD 2005 contest benchmark suite.

In summary, there is a high approximation error between the length of the clique net model and the HPWL, independently of different approaches for the connection weights $w_{x,pq}$. The basic problem of the clique model is that there are connections between inner pins, which contribute to the clique length but which are ignored in the HPWL metric; the HPWL is just the distance between the boundary pins. This problem of the clique net model is demonstrated in Figure 4.3(a). Here, boundary pins are those with the highest or lowest coordinate; all other pins are inner pins. The star net model suffers from the same basic problem as the clique net model: there are two-pin connections, which contribute to the length of the star net, but which are ignored in the HPWL metric.

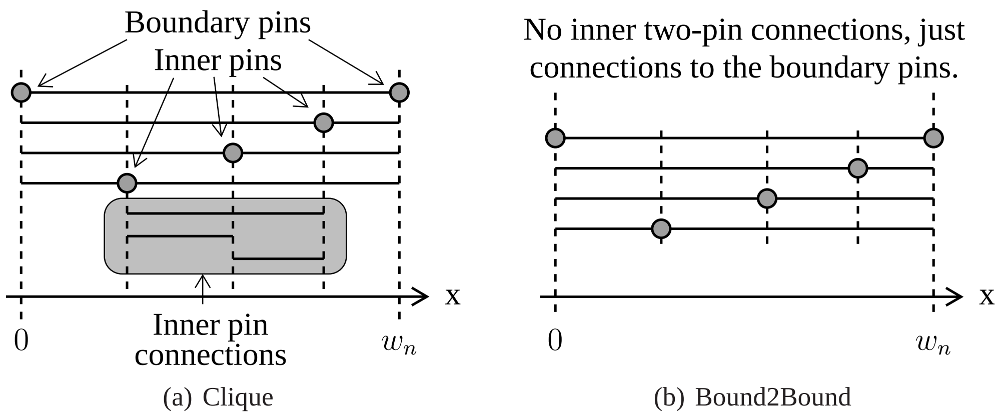


Figure 4.3: Traditional clique net model and the new Bound2Bound net model.

4.2 Bound2Bound Net Model

The new Bound2Bound net model is based on the idea to remove all inner two-pin connections, and to utilize only connections to the boundary pins. An example of a Bound2Bound net model is displayed in Figure 4.3(b). The new net model can be derived from the clique net model. However, its connection weight $w_{x,pq}^{B2B}$ for one two-pin connection is different:

$$w_{x,pq}^{B2B} = \begin{cases} 0 & \text{if pin } p \text{ and pin } q \text{ are inner pins} \\ \dfrac{2}{P-1} \dfrac{1}{|x_p^{pin} - x_q^{pin}|} & \text{else} \end{cases} \tag{4.8}$$

With this connection weight, the quadratic cost function (4.3) of the net is exactly the HPWL in x-direction:

$$\begin{aligned} \Gamma_{n,x} &= \frac{1}{2} \sum_{p=1}^{P} \sum_{q=p+1}^{P} w_{x,pq}^{B2B} (x_p^{pin} - x_q^{pin})^2 && (4.9) \\ &= \frac{1}{2} \frac{2}{P-1} \left[\left| x_1^{pin} - x_2^{pin} \right| + \sum_{q=3}^{P} \left| x_1^{pin} - x_q^{pin} \right| + \sum_{q=3}^{P} \left| x_2^{pin} - x_q^{pin} \right| \right] && (4.10) \\ &= \frac{1}{P-1} \left[w_n + (P-2)\, w_n \right] && (4.11) \\ &= w_n && (4.12) \end{aligned}$$

In (4.10), the linearization $1/|x_p^{pin} - x_q^{pin}|$ is multiplied with the quadratic distance $(x_p^{pin} - x_q^{pin})^2$, which gives the linear distance $|x_p^{pin} - x_q^{pin}|$. Furthermore, all possible two-pin connections are separated in a connection between the two boundary pins ($p = 1, q = 2$), in connections between the “left” boundary pin 1 and inner pins ($p = 1, q \geq 3$), and in connections between the “right” boundary pin 2 and inner pins ($p = 2, q \geq 3$). The inner two-pin connections ($p \geq 3, q > 3$) are not considered as they have a connection weight of zero (4.8). With $w_n = \left| x_1^{pin} - x_2^{pin} \right|$, (4.11) is given. At last, (4.12) expresses that the quadratic cost function is exactly the HPWL in x-direction w_n. Using similar operations for the y-direction, in can be shown that the Bound2Bound net model represents exactly the HPWL in the cost function Γ_n of each net. Thus, the approximation error is zero in the Bound2Bound net model (independently of the number of pins per net), which is shown in Figure 4.2.

4.3 Comparison

With P the number of pins in one net, the clique net model results in $0.5 \cdot P \cdot (P-1)$ two-pin connections. In the star net model, there are P two-pin connections. The new Bound2Bound net model gives $2 \cdot (P-2) + 1$ two-pin connections. Hence, for a two-pin net, the star net model has the most two-pin connections, and the clique net model has the same number of two-pin connections as the Bound2Bound net model. In a three-pin net, all three net models are equivalent in the number of two-pin connections. For all other nets, the clique net model has the most two-pin connections — with a complexity of $O(P^2)$. The Bound2Bound net model has a linear complexity in the number of two-pin connections, and has more two-pin connections than the star net model.

In an average circuit, most of the nets have two or three pins, and nets with lots of pins are rare. Based on such a circuit, the number of two-pin connections is about 75% lower in the Bound2Bound net model than in the clique net model. The runtime for minimizing the quadratic cost function Γ depends mainly on the numbers of two-pin connections and the numbers of pins. Considering the characteristics of the clique and the star net model, there is a trade-off between both net models in an average circuit [Eis99]. For small nets (nets with a small number of pins), the clique net model is better, as no additional star pins are necessary here. For big nets, the star net model is better, as the number of two-pin connections is lower here. The disadvantage of increasing the number of pins with the additional star pins is accepted here, because there are just a few big nets in an average circuit. Compared to such a hybrid usage of the clique model and the star net model, the number of two-pin connections is about the same as in the Bound2Bound net model. However, no additional star pins are introduced in the Bound2Bound net model.

Table 4.1 shows experimental results comparing the Bound2Bound net model with the hybrid clique/star net model. The results represent legal placements, and are obtained with placer “Kraftwerk”. Kraftwerk is described in the next chapters. In the hybrid clique/star net model, GordianL’s (4.4) and Eisenmann’s (4.7) approach for the two-pin connection weights are used. To obtain the best CPU times for the hybrid clique/star net model, all nets with up to six pins are modeled as cliques; the remaining nets are modeled as stars. The new Bound2Bound net model offers the best results in HPWL and CPU time. Eisenmann’s approach increases the HPWL by about 8%, and the CPU time by about 10%. Using GordianL’s approach, the HPWL is increased by about 7%, and the CPU time is increased by about 17%. The Bound2Bound net model has the best HPWL, because it models accurately the HPWL in the quadratic cost function. The Bound2Bound net model has the lowest CPU time, because no additional star pins are used here.

	Bound2Bound		GordianL		Eisenmann	
Circuit	HPWL	CPU	HPWL	CPU	HPWL	CPU
adaptec1	82.43	262	87.96	303	87.63	321
adaptec2	92.85	349	99.63	403	98.54	385
adaptec3	227.22	713	239.97	852	239.05	745
adaptec4	199.43	709	212.31	829	213.32	721
bigblue1	97.67	407	104.81	484	107.23	441
bigblue2	154.74	559	165.27	590	165.60	606
bigblue3	343.32	2070	370.00	2367	389.58	2220
bigblue4	852.40	4147	942.06	5491	958.44	4758
Average	**1.000**	**1.00**	**1.073**	**1.17**	**1.084**	**1.10**

Table 4.1: Comparison between the new Bound2Bound net model and two approaches (GordianL and Eisenmann) for the connection weights in a clique/star net model. Results are normalized to the Bound2Bound net model.

4.4 Approximation Error depending on Module Movement

In quadratic placement, a net model is used at the start of each placement iteration to represent the netlength in the quadratic cost function Γ. To linearize the quadratic length, the net model utilizes the connection weights $w_{x,pq}$. There, $w_{x,pq}$ depends on the pin positions, and thus on the module positions. After the connection weights are determined, the quadratic cost function is minimized by numerical optimization, and the modules are moved to the minimum. During minimization, i.e., during the module movement, the connection weights are not changed. Consequently, there is an inherent approximation error $\hat{\epsilon}$ between the quadratic cost function Γ and the HPWL at the end of each placement iteration. ϵ is the approximation error at the start of the placement iteration, i.e., right at the point where the net model is applied. Based on the statements in the previous section, $\epsilon = 0$ in the Bound2Bound net model.

Figure 4.4 shows the change in the approximation error $\Delta\epsilon = |\epsilon - \hat{\epsilon}|$ depending on the average module movement μ, and three approaches: the Bound2Bound net model, and using the hybrid clique/star net model with GordianL's and Eisenmann's approach for the connection weights. An exact definition of μ is given with (5.25) in the next chapter. Figure 4.4 demonstrates that in general, $\Delta\epsilon$ increases with the module movement. Moreover, there is no essential difference in the three approaches. Hence, the Bound2Bound net model, which separates the pins in inner pins and in boundary pins based on the pin positions before minimizing the quadratic cost function, does not run into significant problems after the pin positions are changed. In addition, Figure 4.4 demonstrates that the lowest $\Delta\epsilon$, and consequently the best placements, are achieved if the modules are moved as little as possible during each placement iteration. This is of interest in Section 5.8 addressing the quality control.

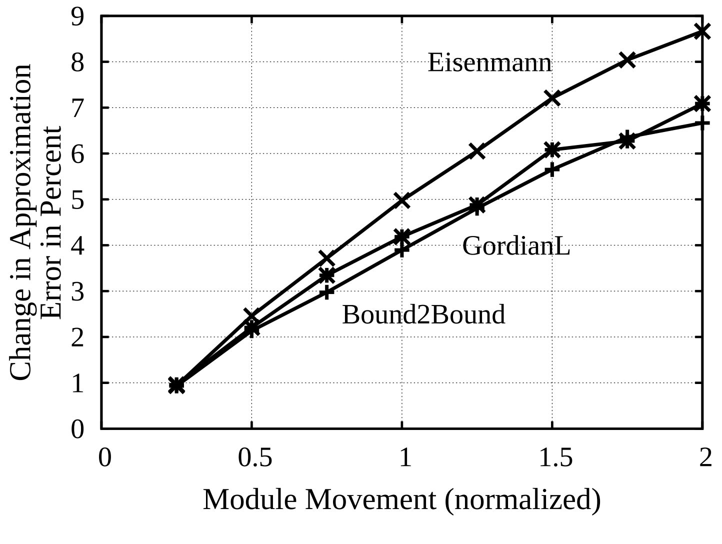


Figure 4.4: Change in approximation error due to module movement for different net models. Results are based on the bigblue1 circuit of the ISPD 2005 contest benchmark suite. Module movement is normalized to the those movement, which gives a good trade-off between runtime and quality (see Section 5.8).

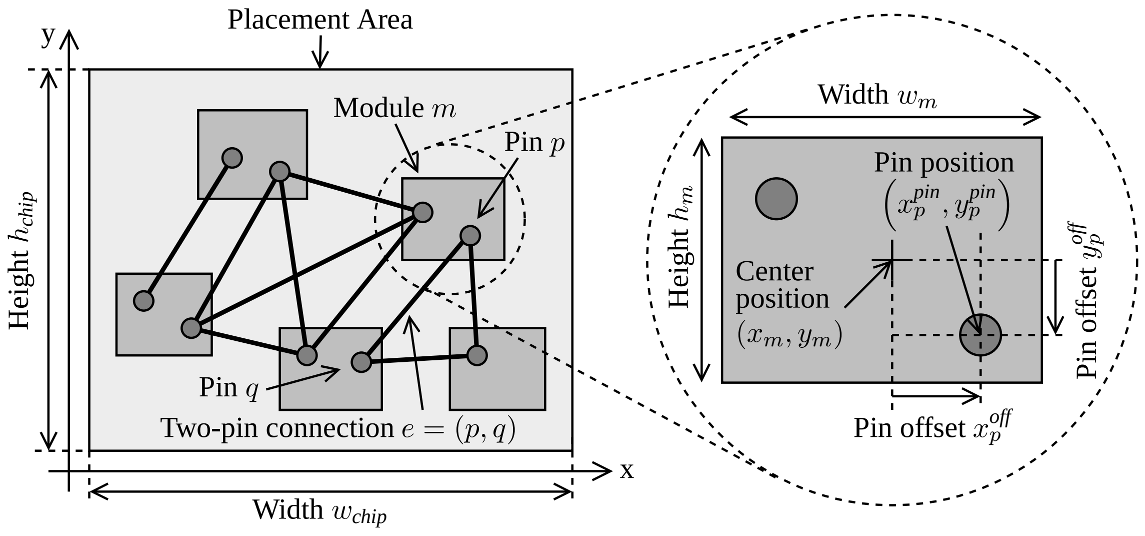


Figure 5.1: Circuit with two-pin connections and different geometrical information.

Chapter 5

Kraftwerk: Force-Directed Quadratic Placement

Before describing the details of Kraftwerk, the basics of quadratic placement are presented first in the chapter.

5.1 Quadratic Placement

Placement in general is based on a gate-level description of the circuit, and quadratic placement in particular is based on that each net is represented by two-pin connections. Figure 5.1 displays a circuit description applicable for quadratic placement. In other words, in quadratic placement, the circuit consists of a set $\mathcal{M}$ of modules, a set $\mathcal{P}$ of pins, and a set $\mathcal{E}$ of two-pin connections. One two-pin connection $e = (p, q) \in \mathcal{E}$ connects pin p with pin q. The set $\mathcal{E}$ of two-pin connections represent the nets, and is obtained by applying a net model to each net of the circuit. Compared to Figure 4(b) of previous section describing net models for quadratic placement, the figure above displays additional geometric information necessary for placement. So, module $m \in \mathcal{M}$ is characterized by its width w_m, its height h_m, and its center position (x_m, y_m). The placement area, i.e., the chip area, is described by its width w_{chip} and its height h_{chip}. Similar to previous section, pin $p \in \mathcal{P}$ is located at position $(x_p^{\mathit{pin}}, y_p^{\mathit{pin}})$.

In quadratic placement, the length of all nets is represented in the quadratic cost function Γ, the sum of the weighted quadratic Euclidean lengths of all two-pin connections:

$$\Gamma = \frac{1}{2} \sum_{e=(p,q)\in\mathcal{E}} w_{x,pq}(x_p^{\mathit{pin}} - x_q^{\mathit{pin}})^2 + w_{y,pq}(y_p^{\mathit{pin}} - y_q^{\mathit{pin}})^2 \tag{5.1}$$

Placement determines the positions of all modules, such that the netlength is minimal. In quadratic placement, the quadratic cost function Γ is minimized. However, Γ depends in (5.1) on the pin positions, and not on the module positions. Hence, a transformation from pin position to module position is necessary. To do this transformation, the function $\pi(p) = m$ maps the pin $p \in \mathcal{P}$ to the module $m \in \mathcal{M}$, according to the relation between module m and pin p:

$$\pi : \mathcal{P} \rightarrow \mathcal{M} \qquad \pi(p) = m\text{: pin } p \in \mathcal{P} \text{ belongs to module } m \in \mathcal{M} \tag{5.2}$$

The pin offset $(x_p^{\mathit{off}}, y_p^{\mathit{off}})$ (see Figure 5.1) describes the difference between the module position and the pin position:

$$x_p^{\mathit{off}} = x_p^{\mathit{pin}} - x_m \quad y_p^{\mathit{off}} = y_p^{\mathit{pin}} - y_m \tag{5.3}$$

Using (5.2) and (5.3), the pin position is described by the pin offset and the corresponding module position:

$$x_p^{\mathit{pin}} = x_{\pi(p)} - x_p^{\mathit{off}} \quad y_p^{\mathit{pin}} = y_{\pi(p)} - y_p^{\mathit{off}} \tag{5.4}$$

Placement also separates the modules in movable and fixed ones, because only the positions of the movable modules have to be determined by placement. The positions of the M movable modules are represented in vector $\mathbf{x}$ for x-direction, and in vector $\mathbf{y}$ for y-direction:

$$\mathbf{x} = (x_1, x_2, x_3, ..., x_M)^T \tag{5.5}$$
$$\mathbf{y} = (y_1, y_2, y_3, ..., y_M)^T \tag{5.6}$$

Using (5.2), (5.4), (5.5), and (5.6), the quadratic cost function Γ represented as a sum in (5.1), can be transformed in a matrix-vector notation:

$$\Gamma = \frac{1}{2}\mathbf{x^T C_x x} + \mathbf{x^T d_x} + \frac{1}{2}\mathbf{y^T C_y y} + \mathbf{y^T d_y} + \mathit{const} \tag{5.7}$$

Matrices $\mathbf{C_x}$ and $\mathbf{C_y}$ represent the connectivity between movable modules, and vectors $\mathbf{d_x}$ and $\mathbf{d_y}$ reflect the connections between movable and fixed modules. Detailed steps to create the matrices and the vectors are described later on. If there are no modules fixed, matrices $\mathbf{C_x}$ and $\mathbf{C_y}$ are positive semidefinite [Hal70]. With some modules fixed, the matrices are positive definite [KV06]. In both cases, Γ is convex, and its minimum is obtained by setting its first derivative to zero. The first derivatives in x- and in y-direction are described by the nabla operators $\boldsymbol{\nabla_x}$ and $\boldsymbol{\nabla_y}$:

$$\boldsymbol{\nabla_x} = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_M}\right)^T \tag{5.8}$$

$$\boldsymbol{\nabla_y} = \left(\frac{\partial}{\partial y_1}, \frac{\partial}{\partial y_2}, ..., \frac{\partial}{\partial y_M}\right)^T \tag{5.9}$$

Using these nabla operators, the derivatives of Γ in x- and y-direction are given:

$$\boldsymbol{\nabla_x}\Gamma = \mathbf{C_x x} + \mathbf{d_x} \tag{5.10}$$

$$\boldsymbol{\nabla_y}\Gamma = \mathbf{C_y y} + \mathbf{d_y} \tag{5.11}$$

Setting these derivatives to zero gives two systems of linear equations:

$$\mathbf{C_x x} + \mathbf{d_x} = \mathbf{0} \tag{5.12}$$

$$\mathbf{C_y y} + \mathbf{d_y} = \mathbf{0} \tag{5.13}$$

Solving these systems with respect to $\mathbf{x}$ and $\mathbf{y}$ gives the module positions $\mathbf{x}$ and $\mathbf{y}$ with minimal netlength. (5.12) and (5.13) demonstrate that $\mathbf{x}$ and $\mathbf{y}$ are determined separately. Moreover, both directions (x and y) or obtained similarly. Hence, the focus is on the x-direction in the following. The y-direction is obtained analogously.

5.2 Creation of Matrix $\mathbf{C_x}$ and Vector $\mathbf{d_x}$

This section describes how matrix $\mathbf{C_x}$ and vector $\mathbf{d_x}$ of the quadratic cost function Γ (5.7) are created. Using (5.2) and (5.3), the cost function in x-direction Γ_x can be written in sum notation, depending on the module positions x_i, $i = 1, 2, ..., M + F$. M is the number of movable modules, and F the number of fixed modules.

$$\Gamma_x = \frac{1}{2} \sum_{e=(p,q)\in\mathcal{E}} w_{x,pq}(x_{\pi(p)} - x_p^{\mathit{off}} - x_{\pi(q)} + x_q^{\mathit{off}})^2 \tag{5.14}$$

The cost of one two-pin connection is given by:

$$\Gamma_{x,pq} = \frac{w_{x,pq}}{2}\left(x_{\pi(p)} - x_p^{\mathit{off}} - x_{\pi(q)} + x_q^{\mathit{off}}\right)^2 \tag{5.15}$$

With this cost, the sum notation of (5.14) can be rewritten:

$$\Gamma_x = \sum_{e=(p,q)\in\mathcal{E}} \Gamma_{x,pq} \tag{5.16}$$

The matrix-vector notation of Γ_x is:

$$\Gamma_x = \frac{1}{2}\mathbf{x^T C_x x} + \mathbf{x^T d_x} + \mathit{const}. \tag{5.17}$$

Vector $\mathbf{x}$ represents the x-position of the M movable modules (5.5). Matrix $\mathbf{C_x} = [c_{x,ij}]$ is a two-dimensional matrix with M rows and M columns. $c_{x,ij}$ is the entry of $\mathbf{C_x}$ in row i and column j. Vector $\mathbf{d_x} = [d_{x,i}]$ is a column vector with M entries. $d_{x,i}$ is the entry of $\mathbf{d_x}$ in row i.

The creation of $\mathbf{C_x}$ and vector $\mathbf{d_x}$ is described at best by using the derivative of Γ_x:

$$\boldsymbol{\nabla}_{\mathbf{x}}\Gamma_x = \mathbf{C_x}\mathbf{x} + \mathbf{d_x} \tag{5.18}$$

A small part of the system of linear equation (5.18) looks like:

$$\begin{pmatrix} \vdots \\ \frac{\partial}{\partial x_i} \\ \vdots \\ \frac{\partial}{\partial x_j} \\ \vdots \end{pmatrix} \Gamma_x = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & c_{x,ii} & \dots & c_{x,ij} & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \dots & c_{x,ji} & \dots & c_{x,jj} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ x_i \\ \vdots \\ x_j \\ \vdots \end{pmatrix} + \begin{pmatrix} \vdots \\ d_{x,i} \\ \vdots \\ d_{x,j} \\ \vdots \end{pmatrix} \tag{5.19}$$

The i-th row in this system of linear equations (5.19) represents the derivative of Γ_x with respect to x_i. In the sum notation (5.16), this derivative is:

$$\frac{\partial}{\partial x_i}\Gamma_x = \sum_{e=(p,q)\in\mathcal{E}} \frac{\partial}{\partial x_i}\Gamma_{x,pq} \tag{5.20}$$

Depending on i, p, and q, the derivative of the cost of $\Gamma_{x,pq}$ of one two-pin connection $e = (p, q)$ is:

$$\frac{\partial}{\partial x_i}\Gamma_{x,pq} = \begin{cases} w_{x,pq}(x_i - x_p^{\mathit{off}} - x_{\pi(q)} + x_q^{\mathit{off}}) & \text{if } i = \pi(p) \\ -w_{x,pq}(x_{\pi(q)} - x_p^{\mathit{off}} - x_i + x_q^{\mathit{off}}) & \text{if } i = \pi(q) \\ 0 & \text{else} \end{cases} \tag{5.21}$$

Using all of this, the contribution of one two-pin connection $e = (p, q)$ to the matrix $\mathbf{C_x}$ and vector $\mathbf{d_x}$ is as follows (with the substitution $i = \pi(p)$ and $j = \pi(q)$):

1. $i, j \leq M$, which means that both modules i and j are movable.
   The diagonal entries $c_{x,ii}$ and $c_{x,jj}$ of the matrix are increased by $w_{x,pq}$, and the off-diagonal entries $c_{x,ij}$ and $c_{x,ji}$ are decreased by $w_{x,pq}$. The entry $d_{x,i}$ of vector $\mathbf{d_x}$ is increased by $w_{x,pq}(-x_p^{\mathit{off}} + x_q^{\mathit{off}})$, and the entry $d_{x,j}$ is decreased by $w_{x,pq}(-x_p^{\mathit{off}} + x_q^{\mathit{off}})$.

2. $i \leq M \wedge j > M$, which means that module i is movable and j is fixed.
   The entry $c_{x,ii}$ of the matrix $\mathbf{C_x}$ is increased by $w_{x,pq}$. In the vector $\mathbf{d_x}$, the entry $d_{x,i}$ is increased by $w_{x,pq}(-x_p^{\mathit{off}} - x_{\pi(q)} + x_q^{\mathit{off}})$.

3. $i > M \wedge j \leq M$, which means that module i is fixed and j is movable.
   The entry $c_{x,jj}$ of the matrix $\mathbf{C_x}$ is increased by $w_{x,pq}$. In the vector $\mathbf{d_x}$, the entry $d_{x,j}$ is decreased by $w_{x,pq}(x_{\pi(p)} - x_p^{\mathit{off}} + x_q^{\mathit{off}})$.

4. $i > M \wedge j > M$, which means that both modules i and j are fixed.
   Matrix $\mathbf{C_x}$ and vector $\mathbf{d_x}$ do not change.

To create matrix $\mathbf{C_x}$ and vector $\mathbf{d_x}$, both are initialized with zeros first. Then, the contribution of each two-pin connection $e \in \mathcal{E}$, as described above, is considered in $\mathbf{C_x}$ and $\mathbf{d_x}$. This gives the matrix $\mathbf{C_x}$ and the vector $\mathbf{d_x}$.

Based on the creation of the matrix $\mathbf{C_x}$, different properties of $\mathbf{C_x}$ can be deduced:

1. The matrix $\mathbf{C_x}$ is symmetric.
2. The diagonal entries of matrix $\mathbf{C_x}$ are all non-zeros, and are all positive.
3. The off-diagonal entries of matrix $\mathbf{C_x}$ are mostly zeros, and if not, they are negative.
4. The matrix is weak diagonal dominant, i.e., for all $i = 1, ..., N$: $\sum_{j=1 \wedge j \neq i}^{N} |c_{x,ij}| \leq |c_{x,ii}|$.
5. Using the Bound2Bound net model, the number of non-zeros depends about linearly on the number of movable modules N. Hence, the matrix $\mathbf{C_x}$ is highly sparse. This property was analyzed using different circuits of various benchmark suites.

Because of these properties, the system of linear equation (5.12) can be solved very efficiently by numerical approaches, e.g., with the conjugate-gradient approach [You03]. Thus, the module positions are determined in low runtime, which is a main advantage of quadratic placement, compared to other placement approaches like non-linear placement or min-cut placement. Details of solving a system of linear equations are presented in Section 5.12.2.

5.3 Force-directed Quadratic Placement

In quadratic placement, the cost $\Gamma_{x,pq}$ (5.15) of one two-pin connection $e = (p, q)$ can be interpreted as the energy of an elastic spring, which is spanned between both pins p and q. In other words, each two-pin connection corresponds to one spring. All two-pin connections of one circuit create a spring system, whose total energy is the quadratic cost function Γ_x (5.16). Since the derivative of the energy with respect to x (or y) is the force in x (or y) direction, the derivative of Γ_x is called the “net” force:

$$\mathbf{F_x^{net}} = \boldsymbol{\nabla_x} \Gamma_x = \mathbf{C_x x} + \mathbf{d_x} \tag{5.22}$$

The name “net” force is because this force is created by the two-pin connections, and the two-pin connections represent the nets. The net force is set to zero in (5.12) and (5.23), to obtain the equilibrium state of the spring system, i.e., the state with minimal energy. This corresponds to the placement with minimal netlength.

$$\mathbf{F_x^{net}} = \mathbf{0} \tag{5.23}$$

With just the net force acting on the modules, the modules are strongly attracted, which results in a lot of module overlap. Mostly, the modules are concentrated in the center of the chip. This is displayed in Figure 5.2 (a). Force-directed quadratic placers utilize an additional force to spread the modules on the chip, and this is done in a sequence of placement iterations. Two placement iterations are shown in Figure 5.2 (b) and (c).

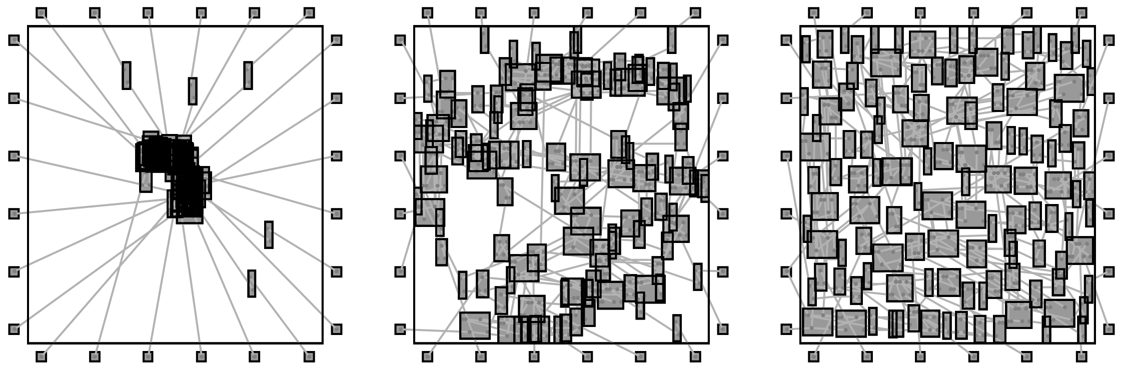

(a) Initial placement with minimal netlength (b) After iteration 5 (c) After iteration 10

Figure 5.2: Placement with minimal netlength (a), and placements at certain placement iterations (b) and (c). In each placement iteration, forces are applied to move the modules and to reduce the overlap.

Previous sections described quadratic placement in general. The following sections presents the novel force-directed quadratic placement approach called “Kraftwerk”. Kraftwerk is based on separating the additional force into two fundamental forces, and both forces are implemented in a systematic way. The result of Kraftwerk’s systematic force implementation is an advanced convergence for various circuits, even for hard instances of macro cell circuits, where the placement area provides only few free space. In other words, Kraftwerk can place many different, and sometimes challenging circuits. Thus, it is a robust placer. Later on, the convergence will be analyzed in theory and based on experimental results. Since Kraftwerk needs only a few placement iterations to spread the modules on the placement area, Kraftwerk is a fast placer.

5.4 Geometry

Before going into details on Kraftwerk’s force implementation, some geometric properties are described now. They are of interest, because they are used frequently in the remaining thesis. The geometric properties of one i module is shown in Table 5.1.

(x'_i, y'_i)	Position at the start of a placement iteration
(x_i, y_i)	Position at the end of a placement iteration
$(\Delta x_i, \Delta y_i)$	Change of the position $\Delta x_i = x_i - x'_i$ $\Delta y_i = y_i - y'_i$
w_i, h_i	Width, height
$A_{mod,i} = w_i \cdot h_i$	Area
$d_{mod,i}$	Individual density, used in the module demand

Table 5.1: Properties of one module i. Position means the center position of the module.

The total module area $A_{\mathit{mod,tot}}$ is the sum of the areas of all M movable and F fixed modules.

$$A_{\mathit{mod,tot}} = \sum_{i=1}^{M+F} A_{\mathit{mod,i}} \tag{5.24}$$

The average module movement μ is:

$$\mu = \frac{1}{M} \sum_{i=1}^{M} \left| (\Delta x_i, \Delta y_i)^T \right| \tag{5.25}$$

Here, $|\cdot|$ means the Euclidean norm.

Table 5.2 summarizes the geometric properties of the chip. Here, it should be noted that the term “chip” and “placement area” are used interchangeably in this thesis.

$(x_{\mathit{chip}}, y_{\mathit{chip}})$	Position of the lower left corner
$w_{\mathit{chip}}, h_{\mathit{chip}}$	Width, height
$A_{\mathit{chip}} = w_{\mathit{chip}} \cdot h_{\mathit{chip}}$	Area

Table 5.2: Properties of the chip.

In the tables above, it is assumed implicitly that the modules and the chip are rectangular. This is done for simplicity. However, in Kraftwerk, the modules and the chip can have any shape, even circles are possible. Assuming rectangular structure, a rectangle function R is suitable to represent the modules and the chip in the two-dimensional space x-y. R is one for all points (x, y) within a rectangle, and zero outside. The rectangle is defined by its lower left corner (x_{ll}, y_{ll}), its width w, and its height h.

$$R(x, y; x_{ll}, y_{ll}, w, h) = \begin{cases} 1 & \text{if } 0 \leq x - x_{ll} \leq w \ \wedge \ 0 \leq y - y_{ll} \leq h \\ 0 & \text{else} \end{cases} \tag{5.26}$$

The rectangle function R can be used to compute different geometrical properties. So, a module distribution $V(x, y)$ is defined by:

$$V(x, y) = \sum_{i=1}^{M+F} R\big(x, y; x'_i - \tfrac{w_i}{2}, y'_i - \tfrac{h_i}{2}, w_i, h_i\big) \tag{5.27}$$

V reflects at point (x, y) the number of module rectangles covering this point. Hence, $V(x, y)$ is the “local module density” at point (x, y). In contrast to this, the term “module density” means the ratio between the total module area $A_{\mathit{mod,tot}}$ and the placement area A_{chip}. The term “module overlap” Ω represents the area $A_\cup$ of the union of all modules, normalized to the total area $A_{\mathit{mod,tot}}$ of all modules:

$$\Omega = 1 - \frac{A_\cup}{A_{\mathit{mod,tot}}} \tag{5.28}$$

$A_\cup$ is determined similar to Klee's measure problem in two dimensions [Kle77]. Based on $V(x,y)$, the area $A_\cup$ of the unions of all modules is calculated by:

$$A_\cup = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \omega(x,y)\,dx\,dy \quad \text{with} \quad \omega(x,y) = \begin{cases} 1 & \text{if } V(x,y) \geq 1 \\ 0 & \text{else} \end{cases} \tag{5.29}$$

If there is no overlap between the modules, then $A_\cup = A_{\mathit{mod,tot}}$, and $\Omega = 0$. If the circuit consists of a high number of modules, all of which are small in dimension and are concentrated somewhere on the chip, then $A_\cup \ll A_{\mathit{mod,tot}}$, and $\Omega \approx 1$.

5.5 One Placement Iteration

Based on one placement iteration, the systematic force implementation of Kraftwerk is described in the following. First, a formal description is given, and then an illustration of the forces is presented. The module positions in each placement iteration are denoted as follows: the vector $\mathbf{x}'$ represents the starting positions, the vector $\mathbf{x}$ represents the new positions, and the vector $\mathbf{\Delta x}$ is the change of position:

$$\mathbf{\Delta x} = \mathbf{x} - \mathbf{x}' \tag{5.30}$$

5.5.1 Move Force

The move force moves the modules in the current placement iteration, in order to reduce the module overlap, and to spread the modules over the chip. To determine the move force, the placement is represented in generic demand-and-supply system D. In principle, the modules create the demand D^{dem}, and the placement area creates the supply D^{sup}.

$$D(x,y) = D^{\mathit{dem}}(x,y) - D^{\mathit{sup}}(x,y) \tag{5.31}$$

The demand-and-supply system has to be balanced, i.e., the integral over the demand has to equal the integral over the supply. This is necessary to adapt the demand completely to the supply.

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} D^{\mathit{dem}}(x,y)\;dx\,dy = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} D^{\mathit{sup}}(x,y)\;dx\,dy \tag{5.32}$$

Using the rectangle function R (5.26), the demand of one module i is:

$$D^{\mathit{dem}}_{\mathit{mod},i}(x,y) = d_{\mathit{mod},i} \cdot R\big(x,y;x'_i - \tfrac{w_i}{2}, y'_i - \tfrac{h_i}{2}, w_i, h_i\big) \tag{5.33}$$

The module demand $D^{\mathit{dem}}_{\mathit{mod}}$ for all M movable and F fixed modules is the sum of all single module demands $D^{\mathit{dem}}_{\mathit{mod},i}$:

$$D^{\mathit{dem}}_{\mathit{mod}}(x,y) = \sum_{i=1}^{M+F} D^{\mathit{dem}}_{\mathit{mod},i}(x,y) \tag{5.34}$$

For simplicity, the individual module density $d_{mod,i}$ is set to one here. Hence, there is no difference between the module demand $D^{dem}_{mod}(x, y)$ and the module distribution $V(x, y)$ (5.27). Section 5.10 presents an advanced approach for scaling $d_{mod,i}$, in order to remove unwanted halos around large each modules. Here, "halo" means free space (see Figure 5.7). In the module demand (5.34), there is no fundamental difference between small or large modules, or between fixed or movable modules. Thus, it can be used to place various circuit types like standard-cell circuits with millions of small modules, mixed-size circuits with small and big modules, and circuits with fixed modules.

Besides a module demand, a module supply D^{sup}_{mod} is necessary for the demand-and-supply system. In the simplest case, the whole placement area provides supply for the modules:

$$D^{sup}_{mod}(x, y) = d_{sup} \cdot R(x, y; x_{chip}, y_{chip}, w_{chip}, h_{chip}) \tag{5.35}$$

The module supply density d_{sup} is determined by (5.34) and (5.32):

$$d_{sup} = \frac{\sum_{i=1}^{M+F} d_{mod,i}\ A_{mod,i}}{A_{chip}} \tag{5.36}$$

Using (5.35), the modules are spread over the whole placement area. Section 5.11 presents an advanced approach for the module supply, in order to spread the modules according to a user-given module density. With this, the modules are not spread over the whole placement area, but can be placed tightly, which reduces the netlength.

The module demand-and-supply system D_{mod} is the module demand D^{dem}_{mod} minus the module supply D^{sup}_{mod}:

$$D_{mod}(x, y) = D^{dem}_{mod}(x, y) - D^{sup}_{mod}(x, y) = D(x, y) \tag{5.37}$$

To place the modules overlap-free on the chip, the module demand-and-supply system is used for D. However, the generic demand-and-supply system D can be extended by additional demand-and-supply systems. For example, it can be extended by the routing demand-and-supply system, in order to optimize routability during placement. This is described in Section 6.4. D can also be used to optimize the temperature profile of a chip [OJ04b, Obe05].

The generic demand-and-supply system D (5.31), and thus the module demand-and-supply system D_{mod} (5.37), is interpreted as a charge distribution, and the charge distribution creates an electrostatic potential Φ by Poisson's equation:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \Phi(x, y) = -D(x, y) \tag{5.38}$$

Section 5.12.1 gives details on computing the potential Φ. The usage of a potential is similar to Eisenmann's placement approach [EJ98, Eis99, Obe05]. However, there, a "constant" force is used, and the force is accumulated over the placement iterations. In contrast to this, Kraftwerk, as presented in this thesis, models the move force with target points and spring connections. Consequently, the move force depends on $\mathbf{x}$, and is not a constant force. In addition, a hold force is used in Kraftwerk, in order to decouple each placement iteration from the previous one. Consequently, no force accumulation is necessary. As a result of this new force modeling, the placement algorithm has an advanced convergence. Section 5.9 analyzes the convergence of Kraftwerk in theory and based on experimental results.

Back to the move force. For module i, this force $F_{x,i}^{move}$ is created by a spring connection between the module and its target point $\mathring{x}_i$.

$$F_{x,i}^{move} = \mathring{w}_i \left(x_i - \mathring{x}_i\right) \tag{5.39}$$

The target point $\mathring{x}_i$ is determined by the starting module position x'_i and the negative gradient of the potential Φ.

$$\mathring{x}_i = x'_i - \frac{\partial}{\partial x}\Phi(x,y)\Big|_{(x'_i,y'_i)} \tag{5.40}$$

Based on the move force, which depends via the target point on the potential Φ (5.39) and (5.40), and the potential represents the demand-and-supply system D (5.38), Kraftwerk is driven by adapting the demand-and-supply system D. $\mathring{w}_i$ in (5.39) is the spring constant of the move force, and is denoted also as the weight of the move force. $\mathring{w}_i$ affects the distance a module i is moved during one placement iteration: with a high $\mathring{w}_i$, the move force of module i pulls a lot on its module, and the module will be moved a long distance. The opposite is true for a small $\mathring{w}_i$. Using target points for the move force, the modules can be moved at most up to their target point during one placement iteration. Hence, the module movement is limited. Moreover, the movement limit is decreasing continuously over the placement iterations. All of this enforces Kraftwerk's convergence. To represent the move force (5.39) in a matrix-vector notation, the weights of the move force are collected in the diagonal matrix $\mathring{\mathbf{C}}_{\mathbf{x}}$:

$$\mathring{\mathbf{C}}_{\mathbf{x}} = diag(\mathring{w}_i) \tag{5.41}$$

The gradients of the potential are collected in the vector $\mathbf{\Phi}_{\mathbf{x}}$:

$$\mathbf{\Phi}_{\mathbf{x}} = \left(\frac{\partial}{\partial x}\Phi\Big|_{(x'_1,y'_1)}, \frac{\partial}{\partial x}\Phi\Big|_{(x'_2,y'_2)}, \ldots, \frac{\partial}{\partial x}\Phi\Big|_{(x'_M,y'_M)} \right)^T \tag{5.42}$$

All target points are represented in the vector $\mathring{\mathbf{x}} = \mathbf{x}' - \mathbf{\Phi}_{\mathbf{x}}$. Therefore, the move force $\mathbf{F}_{\mathbf{x}}^{\mathbf{move}}$ in matrix-vector notation is:

$$\mathbf{F}_{\mathbf{x}}^{\mathbf{move}} = \mathring{\mathbf{C}}_{\mathbf{x}} \left(\mathbf{x} - \mathring{\mathbf{x}}\right) \tag{5.43}$$

5.5.2 Hold Force

To spread the modules iteratively on the chip, the move force is used. However, besides the move force, the net force is acting on the modules to minimize the netlength. Thus, the net force has to be compensated at the start of each placement iteration. Otherwise, the modules collapse back to the initial placement, where the netlength is minimal, but the modules overlap a lot. The compensation of the net force is done by the hold force, and the hold force $\mathbf{F}_{\mathbf{x}}^{\mathbf{hold}}$ equals the negative net force:

$$\mathbf{F}_{\mathbf{x}}^{\mathbf{hold}} = -\left(\mathbf{C}_{\mathbf{x}}\mathbf{x}' + \mathbf{d}_{\mathbf{x}}\right) \tag{5.44}$$

Using only the hold force as one additional force, the modules will not collapse back, but stay at their position in the current placement iteration. In other words, the change in module position $\mathbf{\Delta x}$ is zero. This can be shown by: $\mathbf{F}_\mathbf{x}^\mathbf{net} + \mathbf{F}_\mathbf{x}^\mathbf{hold} = \mathbf{0} \Leftrightarrow \mathbf{C}_\mathbf{x}\mathbf{\Delta x} = \mathbf{0} \Leftrightarrow \mathbf{\Delta x} = \mathbf{0}$. It should be noted here that the hold force equals the net force only at the start of the placement iteration, where the modules are located at $\mathbf{x}'$. Moreover, the hold force is a constant force, as it does not depend on $\mathbf{x}$.

The result of the hold force is that each placement iteration is decoupled from the previous one. Therefore, the placement algorithm can be restarted at any iteration, and the engineering change order (ECO) is supported best. For example, after gate sizing the circuit, and thus introducing module overlap, the placement process can be restarted from the last placement, in order to remove the introduced module overlap. Hence, the placement process needs not be started from scratch, which saves a lot of runtime. Section 8.2 presents experimental results of the ECO feature of Kraftwerk.

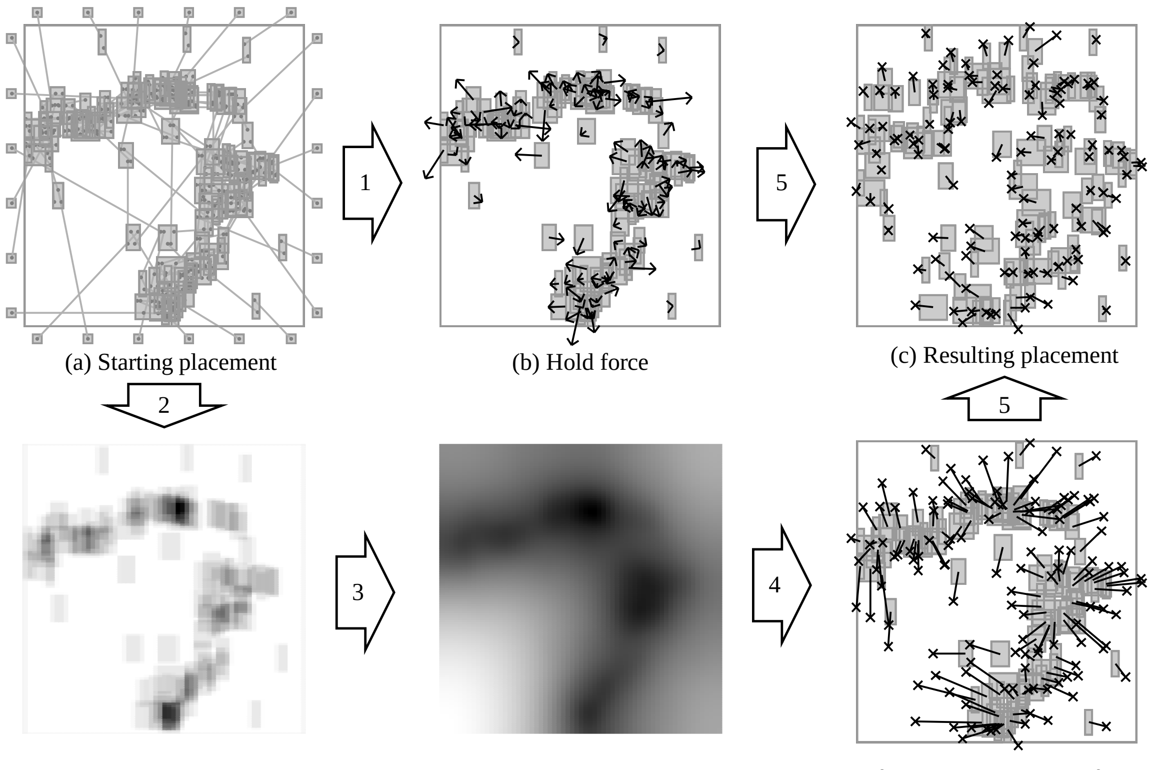


Figure 5.3: Illustration of one placement iteration. The numbers in the big arrows represent the sequence of the steps, taken in each placement iteration. (a) Given placement with modules and nets. (b) Hold force to preserve the placement of (a). (d) Demand-and-supply system. (e) Potential. (d) and (e) are density plots with white color representing low density and black color high density. (f) Move force, created by springs between the modules and their target points. (c) Resulting placement (sum of the net, move, and hold force is zero). The target points are represented by crosses in (c) and (f).

5.5.3 Illustration

Previous sections described in a formal way how the move force and the hold force are determined, and how they are modeled. This section presents an illustration of one placement iteration, particularly of the forces.

The placement iteration starts with a given placement, where each module i is located at (x'_i, y'_i). Figure 5.3 (a) displays such a placement. Ignoring the move force, only the net force is acting on the modules and attracts them together. To compensate for this, the hold force is used, which preserves the given placement. The hold forces are displayed as arrows in Figure 5.3(b).

Based on the module positions, the demand-and-supply D system is created, which represents the local module density. Figure 5.3 (d) shows D of the given placement. D is treated as a charge distribution, which creates an electrostatic potential Φ via Poisson's equation. Such a potential is displayed in Figure 5.3(e). Comparing D in Figure 5.3 (d) with the potential Φ in Figure 5.3 (e) reveals that the potential Φ can be viewed as a smoothed representation of the demand-and-supply system D. Moreover, in regions where D is low, the potential Φ is low, and vice versa.

The gradients of the potential, evaluated at the positions of the modules, determine the target points. The target points are displayed as crosses in Figure 5.3(e). The move force is created by spring connections between the modules and their target points. With the springs to the target points, the modules are moved away from high density regions (black regions in 5.3(d) and (e)) towards low density regions (white regions in 5.3(d) and (e)).

Hence, three forces are acting on the modules in each placement iteration: the net force, the hold force, and the move force. These forces move the modules, until the sum of the forces is zero. The placement, where the sum of all three forces is zero, is the resulting placement of one placement iteration. Figure 5.3(f) displays the resulting placement. Comparing Figure 5.3(c) with (f) shows that the modules are moved towards the target points. In addition, the modules are spread more over the placement area, and the module overlap is reduced.

5.6 Core of Kraftwerk

In summary, three forces are used by Kraftwerk in each placement iteration: the net force $\mathbf{F}_{\mathbf{x}}^{\mathbf{net}}$, and two additional forces: the move force $\mathbf{F}_{\mathbf{x}}^{\mathbf{move}}$ and the hold force $\mathbf{F}_{\mathbf{x}}^{\mathbf{hold}}$. Setting the sum of the three forces to zero (5.45) gives the core system of linear equations (5.46) used in Kraftwerk's iterative placement process.

$$\mathbf{F}_{\mathbf{x}}^{\mathbf{net}} + \mathbf{F}_{\mathbf{x}}^{\mathbf{move}} + \mathbf{F}_{\mathbf{x}}^{\mathbf{hold}} = \mathbf{0} \tag{5.45}$$

$$\left(\mathbf{C}_{\mathbf{x}} + \mathring{\mathbf{C}}_{\mathbf{x}}\right) \mathbf{\Delta x} = -\mathring{\mathbf{C}}_{\mathbf{x}} \mathbf{\Phi}_{\mathbf{x}} \tag{5.46}$$

Solving (5.46) with respect to $\mathbf{\Delta x}$, and updating $\mathbf{x}'$ by $\mathbf{\Delta x}$ gives the new module positions $\mathbf{x}$ in the current placement iteration. Details on solving (5.46) are described in Section 5.12.2. Based on (5.46), Kraftwerk has three degrees of freedom. First, the cost function Γ, represented in $\mathbf{C}_{\mathbf{x}}$. Second, the demand-and-supply system D, represented in $\mathbf{\Phi}_{\mathbf{x}}$. Third, the weights of the move force $\mathring{w}_i$, represented in $\mathbf{\mathring{C}_x}$. Kraftwerk is very flexible and utilizes the degrees of freedom to optimize different objectives (like HPWL netlength and routability), and to control the quality of placement.

5.7 Overview of the Placement Algorithm

Algorithm 1: Global placement algorithm “Kraftwerk”.

```
   // Start with given placement
1  while Module overlap $\Omega \geq 20\%$ do
2      Determine demand-and-supply system $D(x, y)$;
3      Calculate potential $\Phi(x, y)$ based on $D(x, y)$ and Poisson’s equation (5.38);
4      Apply net model;
       // In x-direction (similarly in y-direction):
5      begin
6          Create $\mathbf{C_x}$, $\mathbf{\mathring{C}_x}$, and $\mathbf{\Phi_x}$;
7          Solve $\left(\mathbf{C_x} + \mathbf{\mathring{C}_x}\right)\mathbf{\Delta x} = -\mathbf{\mathring{C}_x}\mathbf{\Phi_x}$ w.r.t. $\mathbf{\Delta x}$;
8          Update module positions $\mathbf{x}$ by $\mathbf{\Delta x}$;
9      end
10     Call quality control;
11 end
   // Next step: final placement (legalization and detailed placement)
```

Algorithm 1 displays the iterative global placement algorithm of Kraftwerk. The global placement starts with a given placement. This can be a placement of a previous run of Kraftwerk, but with additional module overlap introduced, e.g., after gate sizing the placed circuit. Or, the placement is run from scratch, i.e., it is started with the initial placement. For the initial placement, all modules are placed at the center of the chip, and the quadratic cost function Γ (5.7) is minimized over a few iterations (about five). In each iteration, a net model is applied to represent the netlength in Γ.

In global placement, the modules are spread iteratively on the chip. Each placement iteration starts with determining the demand-and-supply system D (line 2), and computing the potential Φ (line 3). Then a net model is applied to determine the weights of the two-pin connections and to represent the netlength in the quadratic cost function Γ (line 4). After that, all elements of the core system of linear equations (5.46) are determined (line 6). Then, (5.46) is solved with respect to $\mathbf{\Delta x}$ (line 7), and the module positions are updated (line 8). These three steps (line 6-8) are done for x- and y-direction. At the end of each placement iteration, a quality control procedure is called, in order to adjust the weights of the move force. The global placement is stopped if the module overlap Ω (5.28) is below a certain limit, e.g., below 20%.

After global placement, final placement is done. Here, the modules are legalized first, which means that the remaining overlap is removed, and the modules are aligned to rows/grid structure if necessary. Considering the remaining overlap of about 20%, the legal placement is obtained quickly (about 5% of the runtime of global placement), and the netlength increase by about 1%. Chapter 7 presents new approaches for legalization. After legalization, detailed placement can be used to improve the legal placement.

5.8 Quality Control

The weights $\mathring{w}_i$ $(i = 1, 2, 3, ..., M)$ of the move force (5.41) are one degree of freedom of Kraftwerk. They are utilized to control the iterative global placement process, and to control the quality of placement. The weight $\mathring{w}_i$ of module i is initialized at the beginning of the global placement process according to:

$$\mathring{w}_i = \frac{A_{mod,i}}{A_{avg}} \cdot \frac{1}{M} \tag{5.47}$$

A_{avg} represents the average module area, and M is the number of movable modules. With the factor $A_{mod,i}/A_{avg}$, the move force (5.39) of module i is proportional to its module area $A_{mod,i}$. Consequently, the big modules are moved faster/further than small modules, and the small modules have to be moved less to obtain an overlap free placement. This improves the netlength, particularly in mixed-size placements, where most of the modules are small, and where most of the nets interconnect small modules.

Based on Rent's rule [LR71], with increasing M, there are more connections between movable modules than connections to fixed modules (e.g., fixed I/O pins). Hence, by minimizing the netlength, the movable modules are more contracted with increasing M. Thus, in the initial placement, the module overlap is higher, and consequently the gradients of the potential Φ are higher. Consequently, the target points (5.40) are farther away from the modules. To preserve the same move force as with small M, the weights of the move force are scaled with $1/M$ in (5.47).

To control the quality during the placement process, the characteristics presented in Section 4.4, and demonstrated in Figure 4.4 are used. There, $\Delta\epsilon$, which is the inherent change in the approximation error between the quadratic cost function Γ and the real objective, depends mainly on the module movement μ. To obtain a high quality placement, i.e., a placement with good netlength, $\Delta\epsilon$ should be as low as possible. Hence, good placements are achieved with a low μ. To control μ, the weights $\mathring{w}_i$ of the move force are used. This is done because with a low $\mathring{w}_i$, the target points attract the modules less, resulting in a low module movement μ. The opposite is true for a high $\mathring{w}_i$. However, with a low μ, a high number of placement iterations are necessary to spread the modules over the chip. Consequently, high quality placements need a high CPU time, and vice versa. Thus, there is a trade-off between quality and runtime, and this trade-off is controlled by the user in setting a target module movement μ_T. The regulation of the module movement μ according to the target movement μ_T is done by the quality control procedure then. This procedure is called at the end of each placement iteration (see Algorithm 1), and is implemented as follows. First, the average movement μ of all modules is calculated. Then, a scale factor κ is determined based on μ and μ_T: if $\mu < \mu_T$, then $\kappa > 1$; if $\mu > \mu_T$, then $\kappa < 1$; else $\kappa = 1$. Figure 5.4 shows a suitable function $\kappa(\mu) = 1 + \tanh(\ln(\mu_T/\mu))$.

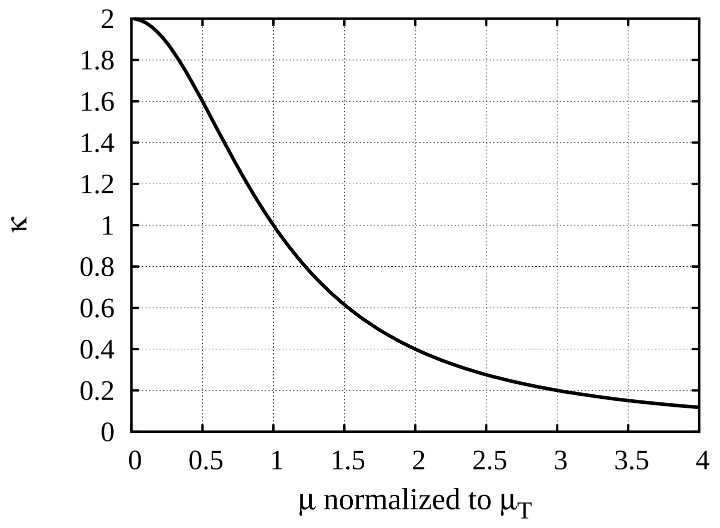


Figure 5.4: Scale factor κ depending on module movement μ and the target module movement μ_T.

After the scale factor κ is determined based on μ and μ_T, the weights $\mathring{w}_i$ of the move force are multiplied with κ:

$$\mathring{w}_i \leftarrow \mathring{w}_i \cdot \kappa \tag{5.48}$$

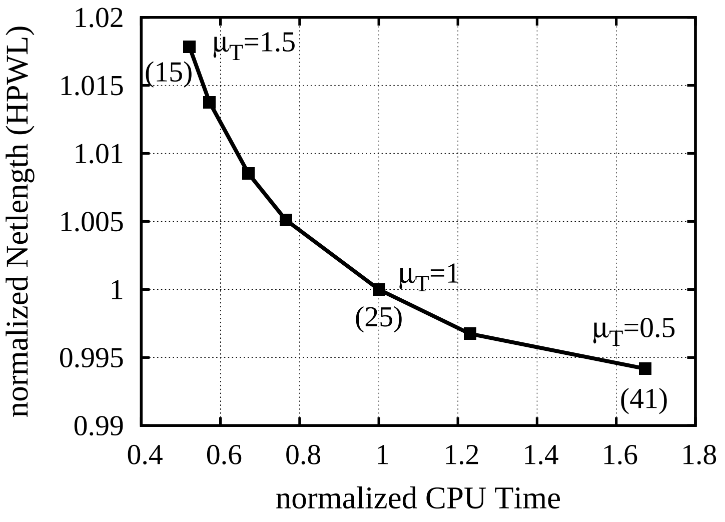


Figure 5.5: Trade-off between runtime and quality based on μ_T. Numbers in brackets represent the number of placement iterations. μ_T is normalized to the average module dimension. Results are based on six circuits of the ISPD 2005 contest benchmark suite.

Figure 5.5 displays the trade-off between runtime (CPU time) and quality (netlength in HPWL). The trade-off is achieved with the presented quality control, and is determined by the user parameter μ_T. With a low μ_T, the number of iterations is high, which results in a high CPU time. Though, the netlength is low then, i.e., the quality is good. With a high μ_T, the CPU time is low, but the netlength is high. To choose a suitable target movement μ_T, the average module dimension is a good reference. The experimental results presented in Section 8 are obtained with μ_T being around this reference.

5.9 Convergence

Kraftwerk is driven by adapting the demand-and-supply system D. Due to the systematic force implementation, the placement algorithm converges such that the demand is adapted further to the supply in each placement iteration. In principle, this means that the module overlap is reduced in each iteration. This section addresses the convergence of Kraftwerk. First, the convergence is analyzed in theory. Then, the convergence is demonstrated by experimental results.

5.9.1 Theory

The following theoretical analysis of the convergence is based on various assumptions. It is intended as a motivation for the presented force implementation. To analyze the convergence in theory, an approximation of the position change Δx_i of module i during one placement iteration is needed first. Since the matrix $\mathbf{C_x}$ is diagonal dominant, it can be approximated with a diagonal matrix $\mathbf{A_x} = diag(\alpha_{x,i})$. Using the Frobenius matrix norm $||E||_F^2 = \sum_{i,j=1}^{N} e_{ij}^2$ and different circuits, the relative error between $\mathbf{C_x}$ and its approximation $\mathbf{A_x}$ is on average about 12%. Hence, the approximation is valid, and the i-th equation of the system of linear equations (5.46) is approximated by:

$$(\alpha_{x,i} + \mathring{w}_i)\,\Delta x_i = -\mathring{w}_i \frac{\partial}{\partial x}\Phi\Big|_{(x_i',y_i')} \tag{5.49}$$

With $\beta_{x,i} = \frac{\mathring{w}_i}{\alpha_{x,i}+\mathring{w}_i}$, (5.49) becomes:

$$\Delta x_i = -\beta_{x,i}\frac{\partial}{\partial x}\Phi\Big|_{(x_i',y_i')} \qquad 0 < \beta_{x,i} \le 1 \tag{5.50}$$

Analogous results are obtained for the y-direction. To make the following formulas simple, it is assumed that $\beta_{x,i} = 1$ and $\beta_{y,i} = 1$. Later on it is described that both variables can have any values of (5.50). The position change $\mathbf{\Delta p}_i$ of module i during one placement iteration is:

$$\mathbf{\Delta p}_i = \begin{pmatrix} \Delta x_i \\ \Delta y_i \end{pmatrix} = -\boldsymbol{\nabla}\Phi\Big|_{(x_i',y_i')} = -\boldsymbol{\nabla}\Phi_i \tag{5.51}$$

$\boldsymbol{\nabla}$ represents the two-dimensional nabla operator $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)^T$.

Now, it is assumed that the demand is created by small elements (e.g., by the modules), the demand elements are moved by the move force, and the change of position of the i-th element is $\mathbf{\Delta p}_i$. The supply is not moved during one placement iteration. Based on these assumptions, the change of demand ΔD_R^{dem} in one region R during one placement iteration is determined next. Like in charge conservation, there is no creation or loss of demand. Thus, the change of demand ΔD_R^{dem} in region R is the flow of the demand across and inside the boundary ∂R during one iteration:

$$\Delta D_R^{dem} = \int_{t_n}^{t_{n+1}} D_{\partial R}^{dem}(t)\, dt \tag{5.52}$$

Each placement iteration can be assigned a certain time step, and the current placement iteration starts at time $t = t_n$ and ends at time $t = t_{n+1}$. The flow of the demand $D_{\partial R}^{dem}(t)$ at time t is created by the demand elements moving inside the region R at time t.

$$D_{\partial R}^{dem}(t) = -\sum_{i \in \partial R} A_i\, d_i\, \mathbf{\Delta p_i}\, \mathbf{n_i} \tag{5.53}$$

The demand element i is defined by the area A_i and the density d_i, and both properties have positive values. According to (5.51), the position change $\mathbf{\Delta p_i}$ of the demand element i is $-\boldsymbol{\nabla}\Phi_i$. The vector $\mathbf{n_i}$ points outside the region R, has a length of one, and is normal to the boundary ∂R. Thus, the product $\mathbf{\Delta p_i}\, \mathbf{n_i}$ represents the normal component of the position change, and is positive if the vector $\mathbf{\Delta p_i}$ points outside the region R. Since the flow inside the region is needed in (5.52), there is a negative sign before the sum in (5.53). Assuming that all vectors $\boldsymbol{\nabla}\Phi$ crossing the boundary ∂R point outside (or inside) in the region R, then (5.53) is finally transformed to:

$$D_{\partial R}^{dem}(t) = \gamma(t) \oint_{\partial R} \boldsymbol{\nabla}\Phi\, d\mathbf{n} \quad \text{with} \quad \gamma(t) \geq 0 \tag{5.54}$$

If some vectors $\boldsymbol{\nabla}\Phi$, which are crossing the boundary ∂R, point outside of the region R and some inside, then further statements on the convergence can only be made if the demand elements are moved an infinite small distance. This would mean that an inifinite high number of placement iterations is necessary to spread the demand over the supply, which results in impracticable runtime. However, in all performed experiments, the demand elements are not moved an infinite small distance, and the convergence to an almost adapted demand-and-supply system is given in about 25 placement iterations. Hence, the assumption about the vectors $\boldsymbol{\nabla}\Phi$ crossing ∂R, made to obtain (5.54), is valid. To obtain $\mathbf{\Delta p_i}$ in (5.51), it was assumed that $\beta_{x,i} = 1$ and $\beta_{y,i} = 1$. If $0 < \beta_{x,i} < 1$ and $0 < \beta_{y,i} < 1$, then $\gamma(t)$ in (5.54) will be smaller, but still non-negative.

Using Poisson's equation (5.38) in the region R with $\int_R \nabla\nabla\Phi\, dx\, dy = -\int_R D\, dx\, dy$, and Gauss' integral theorem $\oint_{\partial R} \mathbf{f} d\mathbf{n} = \int_R \nabla\mathbf{f}\, dx\, dy$, (5.54) yields:

$$D_{\partial R}^{dem}(t) = -\gamma(t) \int_R D\, dx\, dy \quad \text{with} \quad \gamma(t) \geq 0 \tag{5.55}$$

Inserting (5.55) in (5.52) gives the main equation of the convergence analysis:

$$\Delta D_R^{dem} = -\hat{\gamma} \int_R D\, dx\, dy \quad \text{with} \quad \hat{\gamma} = \int_{t_n}^{t_n+1} \gamma(t)\, dt \geq 0 \tag{5.56}$$

The extreme case, where $\hat{\gamma}$ is zero, and hence the demand in one region R does not change, is given for example if the region R is too large. This extreme case is neglected in the following. The main equation of the convergence analysis (5.56) describes that the demand in the region R will decrease during one placement iteration ($\Delta D_R^{dem} < 0$), if the integral of the demand-and-supply in the region is positive ($\int_R D\,dx\,dy > 0$). According to (5.31), this integral is positive, if the demand is greater than the supply. Therefore, (5.56) describes that the demand will decrease in the region R, if the demand is greater than the supply there. Similarly, the demand in the region will increase, if the demand is smaller than the supply. Consequently, (5.56) expresses that the demand is adapted further to the supply in an (arbitrary) region R during each placement iteration. If the whole placement area is the union of such regions, then the demand of the whole placement is adapted further to the supply during each placement iteration.

Therefore, Kraftwerk converges such that the demand is adapted further to the supply in each iteration. And this convergence is based on using a Poisson potential Φ (5.38), target points (5.40), and a constant hold force (5.44).

5.9.2 Experimental Results

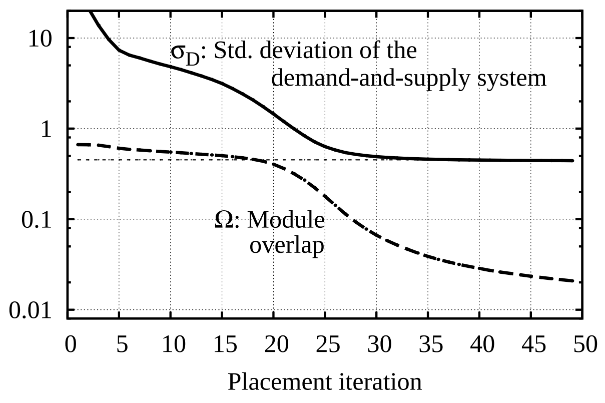
The previous section analyzed the convergence based on different assumption. However, these assumptions may not always be fulfilled in reality. But numerous experiments on sets of different benchmark suites revealed that Kraftwerk converges also in practice. Hence, Kraftwerk is robust and it can place various circuits. Figure 5.6 represents the results of one typical experiment. Here, a circuit with 0.2 million small movable modules and some big fixed modules is placed over a few placement iterations.

The standard deviation σ_D of the demand-and-supply system D, as displayed in Figure 5.6(a), is a suitable metric for the convergence:

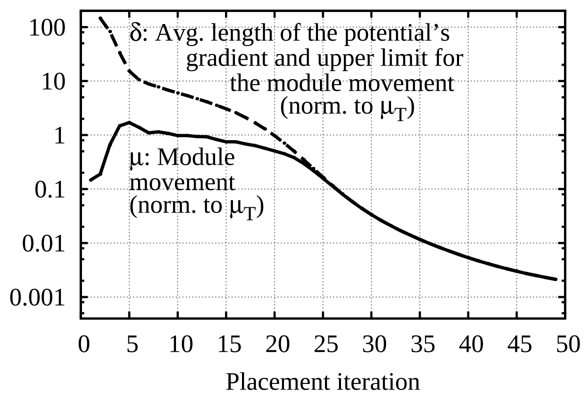
$$\sigma_D^2 = \frac{1}{A_{chip}} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \left(D(x,y) - \mu_D\right)^2 \, dx\,dy \tag{5.57}$$

The lower σ_D is, the better is the adaption of the demand to the supply. Since Kraftwerk adapts the demand further to the supply in each iteration, σ_D should decrease continuously over the iterations. Exactly this effect is illustrated in Figure 5.6(a). A_{chip} in (5.57) represents the area of the chip. The average μ_D of the demand-and-supply system is by definition (5.32) zero.

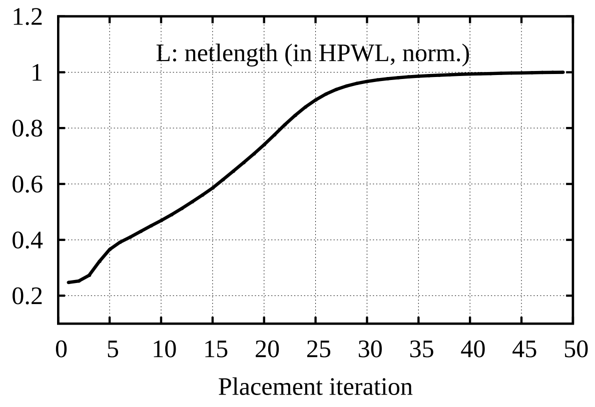
Figure 5.6(a) shows also that σ_D is bound from below, and this lower bound is almost reached at iteration 25. This means, the demand-and-supply system is almost adapted there. The lower bound of σ_D can be computed by assuming that all modules are placed overlap-free on the chip. If the densities d_i of all modules equal the supply density, then the lower bound is zero. Otherwise, the lower bound of σ_D is greater than zero. The circuit represented in Figure 5.6 (a) has a supply density of about 0.45, and almost all modules have a density of 1. This results in a lower bound of σ_D of about 0.45. If the module demand-and-supply is represented in D, then with a decrease of σ_D, the module overlap Ω (5.28) is also decreasing continuously. This behavior of Ω is demonstrated in Figure 5.6(a). Moreover, Ω is about 2% at the last iteration.



(a)



(b)



(c)

Figure 5.6: Demonstration of Kraftwerk's convergence based on the smooth and continuous progress of some characteristic parameters. Circuit: adaptec1 of the ISPD 2005 contest benchmark suite.

The parameter δ, displayed in Figure 5.6(b), represents the average length of the potential's gradient Φ. There, $|\cdot|$ means the Euclidean norm.

$$\delta = \frac{1}{M} \sum_{i=1}^{M} \left| \boldsymbol{\nabla} \Phi(x,y) \,|_{(x_i', y_i')} \right| \tag{5.58}$$

The continuously decreasing standard deviation σ_D of the demand-and-supply system D reflects that the peaks in D are reduced more and more. As the potential Φ represents D by Poisson's equation (5.38), the average length δ of the potential's gradient is also decreasing continuously, as displayed in Figure 5.6(b).

Using (5.50), the module movement of module i in x-direction is limited by the gradient of the potential in x-direction:

$$|\Delta x_i| \leq \left| \frac{\partial}{\partial x} \Phi(x,y) \,\Big|_{(x_i', y_i')} \right| \tag{5.59}$$

Thus, the average module movement μ is limited by δ:

$$\mu = \frac{1}{M} \sum_{i=1}^{M} \left| (\Delta x_i, \Delta y_i)^T \right| \leq \delta \tag{5.60}$$

This relation between μ and δ is demonstrated in Figure 5.6(b). Moreover, the progress of μ has three characteristics. μ is small in the first placement iteration as the weights of the target points $\mathring{w}_i$ are initialized with a small value (5.47). Then, μ is increasing and is around the target movement μ_T because of the quality control described in Section 5.8. After placement iteration 20, μ is continuously decreasing as it reached its upper limit δ and δ is continuously decreasing over all placement iterations.

Figure 5.6(c) shows that the netlength L is continuously and steadily increasing up to around placement iteration 20. This is because the module movement μ is almost constant around μ_T in these iterations. Then, L increases with a lower rate and is almost not changing after iteration 30. This is also due to the module movement μ, which is decreasing after iteration 20 and has a very low value after iteration 30.

In summary, Figures 5.6(a), (b), and (c) demonstrate the convergence of Kraftwerk based on the the smooth and continuous progress of some characteristic parameters. Particularly, the parameter σ_D, as a suitable metric for the convergence, is continuously decreasing. In addition, the global placement, as represented in these figures, is stopped at around iteration 25, because the module overlap Ω is below 20% there. Another useful termination criterion is the value of σ_D.

5.9.3 Limitations

Some limitations of the convergence of Kraftwerk should be noted here. First, if two modules are exactly on top of each other, i.e., their module positions (x_i, y_i) are identical, then they must have different adjacent modules. Otherwise, these critically stacked modules are moved always in the same way, and the overlap between them will not be removed. However, in all of the experiments, such critically stacked modules were not detected. Another limitation of Kraftwerk is the number of placement iterations necessary to obtain the completely adapted placement. Such a placement is described by σ_D being equal to its lower bound, which means there is no module overlap. In theory, this number of iterations is infinite. This is because each module i has to move a certain distance λ_i in the whole placement process, in order to remove all module overlap. Though, the module movement is decreasing over the placement iterations (execept the first ones). Hence, the required distance λ_i is only reached in an infinite number of iterations. However, Kraftwerk is a global placer, and it is stopped if the placement is almost adapted, e.g., if the module overlap is below 20%. These almost adapted placements are obtained in about 25 placement iterations.

5.10 Advanced Module Demand

In Section 5.5.1, the individual module density $d_{mod,i}$ of the module demand is set to one for simplicity. Using $d_{mod,i} = 1$ results in a halo, i.e., free space, around each module. Figure

5.7(a) demonstrate such halos, particularly around the large module at the center. This halo is not wanted, because the small modules are "pushed away" form the large module, which increases the netlength. A better placement with no halo around the large module is shown in Figure 5.7(b). This placement is achieved by scaling $d_{mod,i}$ down for large modules. This section describes details about this approach for preventing unwanted halos.

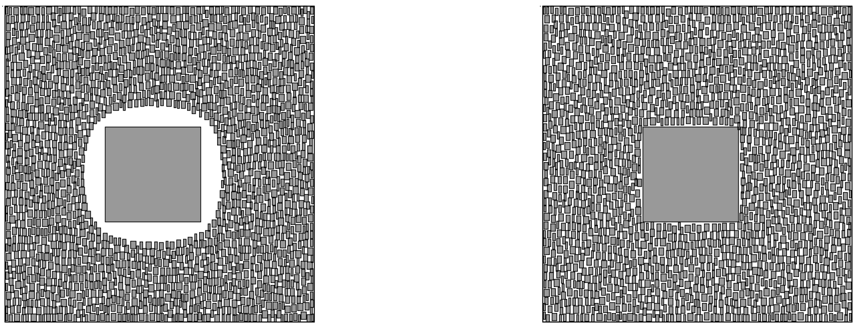

(a) With $d_{mod,i} = 1$: halo around the large module. (b) With scaling down $d_{mod,i}$: no halo.

Figure 5.7: Impact of scaling down the module density $d_{mod,i}$ for large modules. Global placements are displayed here.

The reason for the halos is the potential Φ, and thus the demand-and-supply system D. Section 5.9.1 demonstrates with (5.56) that the demand in a region R will change until the demand equals the supply in this region. Hence, after the global placement iterations, each module i is in an "exclusive" region R_i, and in this region, the demand is balanced by the supply. With a module supply density of d_{sup}, an individual module density $d_{mod,i}$, and a module area $A_{mod,i}$, the exclusive region R_i of module i has the area $A_{R,i}$:

$$A_{R,i} = \frac{d_{mod,i}}{d_{sup}} A_{mod,i} \tag{5.61}$$

In Figure 5.7(a), $d_{sup} = 0.5$ and $d_{mod,i} = 1$. Thus, $A_{R,i} = 2 \cdot A_{mod,i}$, and the exclusive region for the large module in the center is quite big. Consequently, there is a halo around the large module. To prevent the halo, $d_{mod,i}$ has to be scaled down depending on the module area $A_{mod,i}$. A good approach for $d_{mod,i}$ is:

$$d_{mod,i} = \begin{cases} 1 & \text{if } A_{mod,i} < A_{large} \\ \sqrt{\frac{A_{large}}{A_{mod,i}}}\,(1 - d_{sup}) + d_{sup} & \text{else} \end{cases} \tag{5.62}$$

There, the individual module density $d_{mod,i}$ stays one for small modules ($A_{mod,i} < A_{large}$). This conserves the halos around small modules, as these halos are necessary to spread the small modules on the placement area. For large modules ($A_{mod,i} \geq A_{large}$), $d_{mod,i}$ is scaled down with increasing module area $A_{mod,i}$. In addition, $d_{mod,i}$ is bound from below by the supply density d_{sup}. Otherwise, the placement algorithm would not convergence to an overlap-free placement. A good value for the reference area A_{large} used in (5.62) is $50\,A_{avg}$, with A_{avg} denoting the average module area. Figure 5.7(b) demonstrates the result of scaling down the individual module density $d_{mod,i}$ by (5.62). Here, the halo around the large module is removed.

5.11 Advanced Module Supply

In Section 5.5.1, the whole placement area provides supply for the modules. This results in that the modules are spread on the whole placement area, as shown in Figure 5.8 (a). The module density equals the chip utilization u then.

$$u = \frac{A_{\textit{mods,mov}}}{A_{\textit{chip}} - A_{\textit{mods,fixed}}} \tag{5.63}$$

$A_{\textit{mods,mov}}$ is the total area of all movable modules, $A_{\textit{mods,fixed}}$ is the total area of all fixed modules, and $A_{\textit{chip}}$ is the chip area. Prior to placement, the chip area is determined by floorplanning. Thus, the chip area is fixed during placement. To lower the netlength, it may be allowed to pack the modules with a higher density than u. Figure 5.8 (c) demonstrates the effect that with increasing module density, the netlength decreases. This section presents an approach to control the module density.

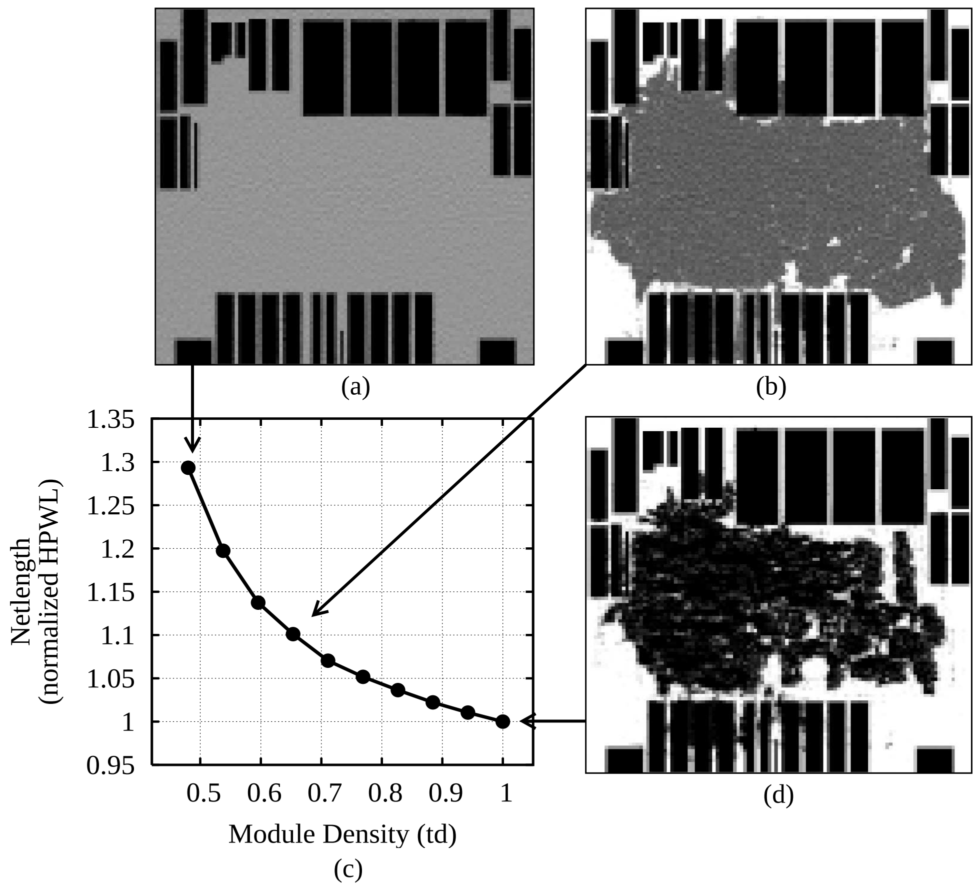


Figure 5.8: Control of the module density. Module density plots (a), (b), and (d) represent a low density with white color and high density with black color. The big black rectangles represent fixed big modules. Based on a circuit with 0.2 million small movable modules and some big fixed modules.

Since Kraftwerk adapts the demand to the supply, and the modules are represented in the demand, the supply can be used to control the module density. Based on an user given module target density td, the creation of the module supply is done in two steps (see Figure 5.9). First, an initial module supply $D^{sup}_{mod,init}(x, y)$ with the value td is created at each point (x, y) where the module demand is greater zero $D^{dem}_{mod}(x, y) > 0$. Second, an additional module supply $D^{sup}_{mod,add}(x, y)$ with the value td is created around the initial module supply. The additional module supply is needed to get a balanced demand-and-supply system (5.32). The sum of the initial and additional module supply gives the module supply: $D^{sup}_{mod} = D^{sup}_{mod,init} + D^{sup}_{mod,add}$. If (5.62) is used for the module demand, then $d_{sup} = td$.

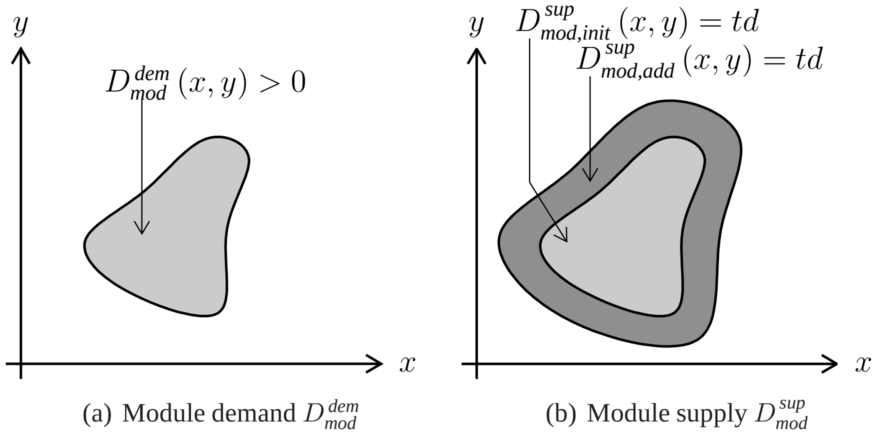

y
$D^{dem}_{mod}(x, y) > 0$
x
(a) Module demand D^{dem}_{mod}
y
$D^{sup}_{mod,init}(x, y) = td$
$D^{sup}_{mod,add}(x, y) = td$
x
(b) Module supply D^{sup}_{mod}

Figure 5.9: Creation of the module supply $D^{sup}_{mod} = D^{sup}_{mod,init} + D^{sup}_{mod,add}$ based on the module demand D^{dem}_{mod}. This controls the module density to be td.

Since the potential is solved numerically, the potential is calculated on a grid structure. The demand-and-supply system is also represented by a grid structure. The grid structure divides the placement area in a number of bins. Hence, the two steps described above to create the module supply can be done by using the bins. First, the initial module supply is created in each bin where the module demand is greater zero. Second, the additional module supply is created iteratively around the bins, where the initial module supply was deposited.

5.12 Implementation Details

This section covers different implementation details about computing the electrostatic potential $\Phi(x, y)$ and how to solve the core system of linear equations (5.46) efficiently.

5.12.1 Calculation of the Potential

The target points (5.40) of the move force are determined by the gradient of the potential $\Phi(x, y)$. The potential $\Phi(x, y)$ itself is given by the demand-and-supply system $D(x, y)$ and Poisson's equation (5.38):

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\Phi(x, y) = -D(x, y) \tag{5.64}$$

One approach to calculate the potential $\Phi(x, y)$ is to use a Greens function $G(x, y) = \ln\left(x^2 + y^2\right)$ in combination with a convolution:

$$\Phi(x, y) = k \cdot D(x, y) \star G(x, y) \quad \text{with} \quad k = \textit{const} \tag{5.65}$$

The convolution can be solved by the Fourier Transformation [Wil80]. For a computer algorithm using numerics, the Discrete Fourier Transformation (DFT) is applicable. This means that the demand-and-supply system D has to be discretized. The discretization is done by overlaying the placement area with a grid structure, resulting in a number of bins. The average value of D in a bin gives the discrete value in this bin. Based on Nyquist-Shannon sampling theorem [Nyq28], the maximal bin dimension has to be half of the minimal module dimension. However, with N the number of modules, the number of bins would be $O(N^2)$, which results in an impracticable computational complexity. Based on experimental results, the bin dimension can be reduced to about the average module dimension, without loss in quality. With this, the number of bins is $O(N)$, thus the computational complexity is practicable, and depends linearly on the number N of modules.

Since the DFT results in periodic functions, the grid structure for discretization needs to be twice as big as the placement area in each direction (x and y). This increases the number of bins, and thus increases the runtime to calculate the potential. A faster numeric approach to compute the potential is to transform the Laplace operator $\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$ into finite differences [HW76]. Here, the demand-and-supply system has to discretized as described above, too. However, the grid structure needs not be enlarged in each direction. Hence, the runtime is lower compared to the DFT approach. Using finite differences, the potential is determined by solving a system of linear equations. There, the system matrix is of special kind, namely it has a band structure. This means that all entries of the matrix are near the diagonal. Such a system of linear equations is solved efficiently by a geometric multigrid method like DiMEPACK [KW01].

5.12.2 Solving the System of Linear Equations

The core of Kraftwerk is to solve the system of linear equations (5.46) with respect to $\mathbf{\Delta x}$ in each placement iteration. Adding $\mathbf{\Delta x}$ to $\mathbf{x}'$ gives the new module positions $\mathbf{x}$ in each placement iteration. By substituting $\mathbf{C_x} + \mathring{\mathbf{C}}_\mathbf{x} = \mathbf{A}$ and $-\mathring{\mathbf{C}}_\mathbf{x}\mathbf{\Phi_x} = \mathbf{b}$, the system of linear equation is:

$$\mathbf{A} \cdot \mathbf{\Delta x} = \mathbf{b} \tag{5.66}$$

The matrix $\mathbf{A}$ has similar properties as the matrix $\mathbf{C_x}$ (see Section 5.2): $\mathbf{A}$ is symmetric, positive definite, and highly sparse. Compared to the matrix, which is used in solving Poisson's differential equation (see previous Section 5.12.1), $\mathbf{A}$ has no band structure, i.e., the off-diagonal entries are not always near the diagonal. Therefore, geometric multigrid methods like DiMEPACK are not applicable to solve the system of linear equations (5.66). An efficient method to solve (5.66) is the conjugate-gradient (CG) approach [You03]. This is an iterative approach, and in each solving iteration, a matrix-vector multiplication $\mathbf{A} \cdot \mathbf{r}$ is executed. The runtime of the CG approach depends mainly on this matrix-vector multiplication, and on the number of solving iterations. The number of solving iterations can be lowered by using preconditioning matrices. An efficient precondition matrix is based on the diagonal entries of $\mathbf{A}$. The runtime of the matrix-vector multiplication depends, amongst others, on the arithmetic precision (single or double precision). Since (5.66) is solved for the change $\mathbf{\Delta x}$ in the module positions, single precision is sufficient. This decreases the runtime of solving (5.66) by a factor of two, compared to double precision, which would be necessary if (5.66) is solved for the absolute module positions $\mathbf{x}$. Amongst others, this single precision arithmetic gives the fast runtimes of Kraftwerk.

Chapter 6

Routability-Driven Placement

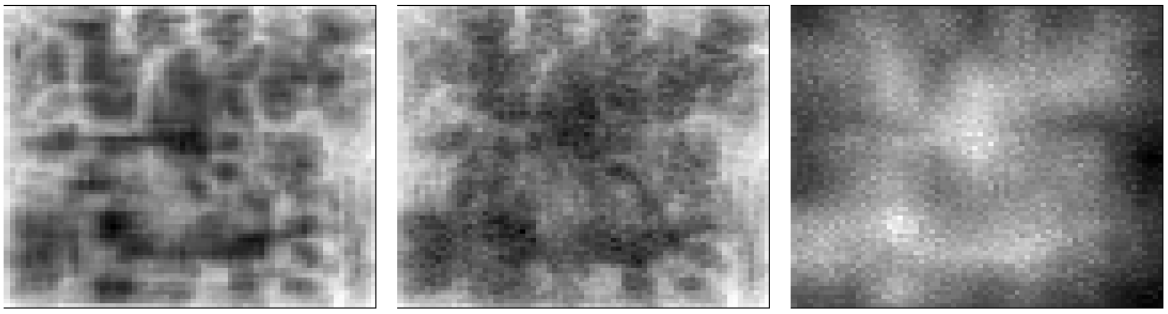

(a) RUDY of a circuit (b) Exact routing demand (c) Module density

Figure 6.1: Routing demand estimation RUDY (a), exact routing demand (b), and module distribution after routability-driven placement (c). White color represents low density, and black color high density. Results are based on the circuit ibm12e of IBM-PLACE 2.0 benchmark suite.

The layout synthesis of a circuit means to place the modules on the chip, and to route the nets, which connect the modules. These two steps (placement and routing) are done consecutively, mostly by different computer algorithms. To obtain the best results, routing must be considered during placement. This is called "routability-driven placement", and this chapter presents new approaches for it. In detail, a circuit may have a high routed wirelength, i.e., a high wirelength after routing, or the circuit may even not be routable, because of "congested regions". Congested regions are regions on the chip, where too much wires are necessary to route the nets, than routing tracks are available there. In other words, in congested regions, the routing demand, created by the nets, exceeds the routing supply, given by the routing layers. Consequently, routability-driven placement means to remove the congestions during placement. To do this, two problems have to be solved. First, an accurate and fast estimation of the routing demand is necessary. The most precise information about the routing demand would be given by routing, but routing can not be performed during the iterative placement process, because of the enormous runtime of routing. Second, the routing demand estimation must be integrated in the placer to optimize routability. This chapter addresses both problems. First, the efficient routing demand estimation called "RUDY" is presented. After that, the integration of the routing demand in the placer Kraftwerk is described.

6.1 RUDY: Efficient Routing Demand Estimation

This section presents RUDY, which is a novel and efficient estimation of the routing demand. First, the routing demand of one net $n \in \mathcal{N}$ is described, and then the routing demand of a complete circuit is presented. In general, one net n consists of several pins. The positions of the pins determine the “net rectangle”, which is the smallest rectangle enclosing all pins. Amongst others, Table 6.1 describes the geometric properties of the net rectangle. This rectangle has the lower left corner located at $(x_{\textit{net,n}}, y_{\textit{net,n}})$, a width of $w_{\textit{net,n}}$, and a height of $h_{\textit{net,n}}$. The product of width and height gives the area $A_{\textit{net,n}}$. Independent of the net rectangle, net n has the wirelength L_n. L_n can be the routed wirelength, i.e., the wirelength after routing. However, routing takes some runtime. To estimate the routing demand in low runtime, it is better to use an estimation of the routed wirelength for L_n. A suitable estimation is the half-perimeter wirelength (HPWL), which is the width $w_{\textit{net,n}}$ plus the height $h_{\textit{net,n}}$ of the net rectangle.

$(x_{\textit{net,n}}, y_{\textit{net,n}})$	Position of the lower left corner
$w_{\textit{net,n}}, h_{\textit{net,n}}$	Width, height
$A_{\textit{net,n}} = w_{\textit{net,n}} \cdot h_{\textit{net,n}}$	Area
L_n	Wirelength

Table 6.1: Properties of one net n, and in particular of its “net rectangle”.

The routing demand estimation technique RUDY is based on the idea to assume a uniform wire density $d_{\textit{wire,n}}$ within the net rectangle of each net. There, the acronym RUDY stands for *R*ectangular *U*niform wire *D*ensit*Y*. The RUDY of one net is displayed in Figure 6.2. In principle, the wire density $d_{\textit{wire,n}}$ of net n is the ratio between the wire area $A_{\textit{wire,n}}$ and the net area $A_{\textit{net,n}}$. The wire area is the product of the wirelength L_n and the wire width p. The wire width p is the average wire-to-wire pitch of process technology, used to fabricate the circuit.

$$d_{\textit{wire,n}} = \frac{A_{\textit{wire,n}}}{A_{\textit{net,n}}} = \frac{L_n \cdot p}{w_{\textit{net,n}} \cdot h_{\textit{net,n}}} \tag{6.1}$$

The routing demand $D^{\textit{dem}}_{\textit{rout,n}}$ of one net n using RUDY is the wire density $d_{\textit{wire,n}}$ inside its net rectangle, and zero outside. Using the rectangle function (5.26), the RUDY of one net is:

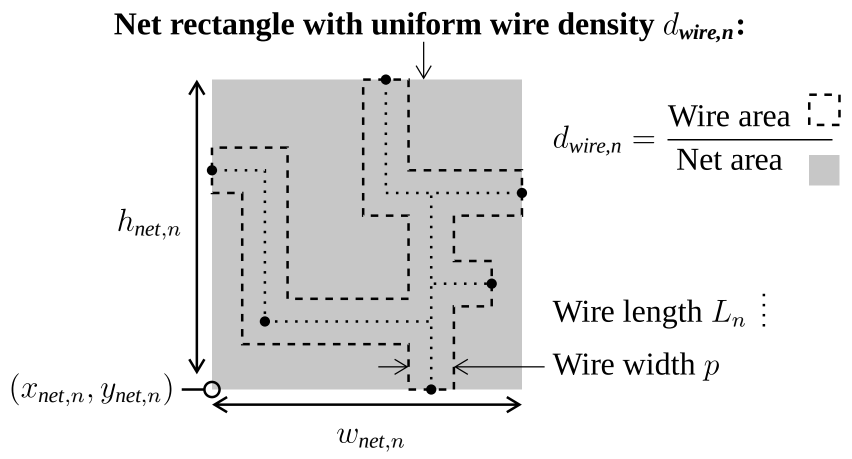
$$D^{\textit{dem}}_{\textit{rout,n}}(x,y) = d_{\textit{wire,n}} \cdot R(x, y; x_{\textit{net,n}}, y_{\textit{net,n}}, w_{\textit{net,n}}, h_{\textit{net,n}}) \tag{6.2}$$

The routing demand $D^{\textit{dem}}_{\textit{rout}}$ of all N nets, i.e., the RUDY of a circuit, is the sum of all net routing demands $D^{\textit{dem}}_{\textit{rout,n}}$:

$$D^{\textit{dem}}_{\textit{rout}}(x,y) = \sum_{n=1}^{N} D^{\textit{dem}}_{\textit{rout,n}}(x,y) \tag{6.3}$$

6.2 Characteristics of RUDY

Figure 6.1 shows different density plots. The exact routing demand, displayed in Figure 6.1(b), is given after routing, and describes at point (x, y) the number of wires, covering



Net rectangle with uniform wire density $d_{wire,n}$**:**
$d_{wire,n} = \frac{\text{Wire area}}{\text{Net area}}$
Wire area
Net area
$h_{net,n}$
$(x_{net,n}, y_{net,n})$
$w_{net,n}$
Wire length L_n
Wire width p

Figure 6.2: Routing demand estimation RUDY of one net n.

this point. The comparison between the estimated routing demand using RUDY (see Figure 6.1(a)) and the exact routing demand (see Figure 6.1(b)) demonstrates that RUDY estimated very precisely regions with high routing demand (high wire density), as well as regions with low routing demand (low wire density).

Figures 6.1 (a) and (b) give a graphic comparison between RUDY and the exact routing demand. In the following, a more precise comparison based on some characteristic parameters is given. Moreover, not only RUDY is compared with the exact routing demand, but also the quality of two other estimation techniques is analyzed, namely the approach called "RISA" [lEC94], and the approach of Westra et al. [WBG04]. To do the comparison, the chip is overlayed with a fine grid structure, which results in a number of bins. In each bin i, the exact routing demand, and the three estimated routing demands are determined. These four routing demands in bin i are represented in $r_{exact}[i]$, $r_{RUDY}[i]$, $r_{RISA[i]}$, and $r_{Westra}[i]$, respectively. For each routing demand, the average $\mu_{<dem>}$ over all N bins is computed:

$$\mu_{<dem>} = \frac{1}{N}\sum_{i=1}^{N} r_{<dem>}[i] \quad \text{with} \quad \textit{dem} = \text{exact, RUDY, RISA, or Westra} \tag{6.4}$$

The average error μ_{Error} for each estimation technique is then given by:

$$\mu_{Error}(\textit{est}) = \mu_{<est>} - \mu_{exact} \quad \text{with} \quad \textit{est} = \text{RUDY, RISA, or Westra} \tag{6.5}$$

Table 6.2 shows that RUDY has the best average error, Westra's approach has a higher average error, and the average error of RISA is far too high.

To obtain the standard deviation of the error σ_{Error}, the routing demand of each estimation technique is scaled such that the average error is zero:

$$r_{<est>}[i] \leftarrow r_{<est>}[i]\,\frac{\mu_{exact}}{\mu_{<est>}} \quad \text{with} \quad \textit{est} = \text{RUDY, RISA, or Westra} \tag{6.6}$$

The standard deviation of the error σ_{Error} is:

$$\sigma^2_{Error}(est) = \frac{1}{N-1}\sum_{i=1}^{N}\left(r_{<est>}[i] - r_{exact}[i]\right)^2 \quad \text{with} \quad est = \text{RUDY, RISA, or Westra} \tag{6.7}$$

In (6.7), the factor $1/(N-1)$ is used instead of $1/N$, in order to address the unbiasedness [And74]. However, in the evaluated circuits, the number N of bins is hundred and above, hence there is no big difference between $1/N$ and $1/(N-1)$.

In Table 6.2, the standard deviation of the error σ_{Error} of the three estimation techniques are all about the same. Westra's approach is the best, RISA is the worst, and RUDY in the middle between them. In the runtime necessary to obtain the routing demand, RUDY is as fast as RISA. Westra's approach needs about 10 times more runtime. To obtain the exact routing demand, i.e., to route the circuit, takes about factor 4000 more runtime.

	RUDY	**RISA**	**Westra et al.**	**Exact**
μ_{Error}	0.013	200653	0.939	—
σ_{Error}	0.144	0.153	0.130	—
CPU	1.00	1.00	10.66	3800.00

Table 6.2: Comparison of RUDY and other approaches to estimate the routing demand. The exact routing demand, as given by routing the circuit, is used as a reference for μ_{Error} and σ_{Error}. Statistic is based on all circuits of the IBM-PLACE 2.0 benchmark suite.

In summary, RUDY is a fast and accurate routing demand estimation approach. In contrast to other approaches, RUDY needs no grid structure. The grid structure used above to determine the estimation error is because RISA and Westra's approach rely on it, and because of numerical reasons: the continuous routing demand of RUDY must be discretized by bins in order to compute the estimation error on a computer. In addition, RUDY does not use a routing model to describe possible routes of each net. Other approaches like Westra's technique are using such routing models, and are fitting the possibilities of the routes to the results obtained by routing. Using such routing models results in a dependency between the routing demand estimation technique and the router. RUDY is not based on a routing model, and thus RUDY estimates the routing demand independently of the router.

6.3 Routing Supply

Besides the routing demand, there is also a routing supply. The routing supply is given by the routing layers of the chip. Based on the rectangle function (5.26), the routing supply is:

$$D^{sup}_{rout}(x, y) = d^{sup}_{rout} \cdot R(x, y; x_{chip}, y_{chip}, w_{chip}, h_{chip}) \tag{6.8}$$

Routing obstacles (e.g., fixed macros) are excluded from the routing supply. The routing supply density d^{sup}_{rout} is determined by considering a balanced demand-and-supply system (5.32).

If there are no routing obstacles, the routing supply density is:

$$d_{\mathit{rout}}^{\mathit{sup}} = \frac{\sum_{n=1}^{N} d_{\mathit{wire},n} \cdot A_{\mathit{net},n}}{A_{\mathit{chip}}} \tag{6.9}$$

6.4 Integration in Kraftwerk

The routing demand $D_{\mathit{rout}}^{\mathit{dem}}$ (6.3) and the routing supply $D_{\mathit{rout}}^{\mathit{sup}}$ (6.8) give the routing demand-and-supply system D_{rout}:

$$D_{\mathit{rout}}(x,y) = D_{\mathit{rout}}^{\mathit{dem}}(x,y) - D_{\mathit{rout}}^{\mathit{sup}}(x,y) \tag{6.10}$$

To drive placement by routability in Kraftwerk, its demand-and-supply system D (5.38) has to be a combination of the module demand-and-supply D_{mod} (5.37) and the routing demand-and-supply D_{rout}:

$$D(x,y) = (1 - w_{\mathit{rout}})D_{\mathit{mod}}(x,y) + w_{\mathit{rout}} D_{\mathit{rout}}(x,y) \tag{6.11}$$

In Kraftwerk, the demand, which is created by the modules and the nets in (6.11) now, is adapted to the supply, which is given by the chip. Therefore, Kraftwerk's approach for routability-driven placement can be viewed as placing the modules and the net rectangles concurrently on the chip.

The routing weight w_{rout} in (6.11) represents the degree of routability optimization: with $w_{\mathit{rout}} = 0$, routability is ignored, and with $w_{\mathit{rout}} = 1$, just routability is optimized, ignoring the placement of the modules. The optimal w_{rout}^{*}, which gives the lowest routed wirelength, depends on the circuit and the router. For one circuit and one given router, w_{rout}^{*} is determined by the golden section search method [Kie53]. This is a numerical optimization method, which evaluates the routed wirelength *rWL* for certain values of w_{rout}, and iteratively refines the interval, in which the minimum of *rWL* is located. The interval is refined by using the golden ratio $(1+\sqrt{5})/2$, in order to have best convergence speed. To evaluate *rWL* for one value of w_{rout}, the circuit is placed with w_{rout}, and the resulting placement is routed.

Figure 6.3 displays the dependency of some parameters on the routing weight w_{rout}. One parameter is the standard deviation of the routing demand σ_{rout}, which is calculated by:

$$\sigma_{\mathit{rout}}^{2} = \frac{1}{A_{\mathit{chip}}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \left(D_{\mathit{rout}}^{\mathit{dem}}(x,y) - \mu_{\mathit{rout}}\right)^{2} dx\, dy \tag{6.12}$$

μ_{rout} is the average value of $D_{\mathit{rout}}^{\mathit{dem}}(x,y)$. Other parameters are the netlength and the routed wirelength. The dependency of these parameters on w_{rout} are as follows:

1. σ_{rout} decreases with increasing routing weight w_{rout}. This means that the peaks in the routing demand are reduced more and more. Thus, also the routing demand in congested regions is reduced.

2. The netlength measured in HPWL or RSMT (rectilinear Steiner minimal tree) increases with increasing routing weight w_{rout}.

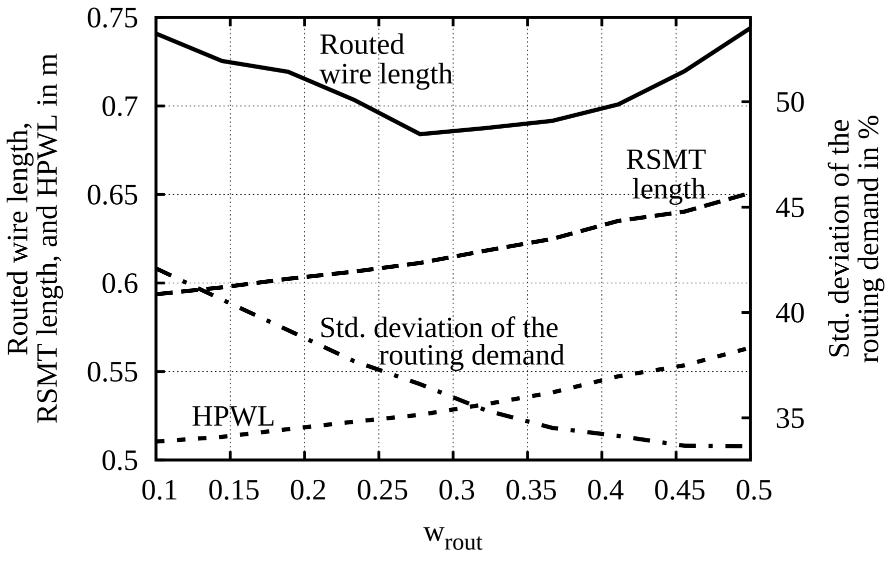


Figure 6.3: Dependency of some parameters on the routing weight w_{rout}. Results are based on ibm01e of the IBM-PLACE 2.0 benchmark suite.

3. There is a trade-off between netlength and σ_{rout}, resulting in an optimal routed wirelength. For the circuit used in Figure 6.3, the optimal w^*_{rout} is 0.28.

Based on these characteristics, which are demonstrated in Figure 6.1, some statements can be derived:

1. Kraftwerk reduces the routing demand in congested regions. This is demonstrated by a decrease of σ_{rout} with increasing w_{rout}.

2. Kraftwerk increases the routing supply in congested regions. This can be shown by comparing the wire density plot in Figure 6.1(b) with the module density plot in Figure 6.1(c). In congested regions, where the wire density is high, the module density is low. Since modules block some routing layers, a low module density in congested region means more routing supply there.

3. The HPWL is an efficient estimation of the routed wire length. This is because the HPWL correlates to the routed wire length as good as the RSMT length does. However, the HPWL is much faster determined than the RSMT [Chu04].

To validate the statement that the HPWL is an efficient estimation of the routed wire length, four estimation techniques for the routed wirelength were tested: HPWL, RSMT (length of rectilinear Steiner tree), RMST (length of minimum spanning tree), and RISA [lEC94] (estimating the length of one net by a function depending on the HPWL and the number of pins). The four different estimation techniques were integrated in RUDY and in the quadratic cost function Γ. The integration in RUDY was done by using the estimated wirelength in L_n. The integration in Γ was done by using the Bound2Bound net model, and scaling the connection weights of each net by the ratio between the estimated wirelength and the HPWL. The results of the four estimation techniques are summarized in Table 6.3, and are based on all circuits of the IBM-PLACE 2.0 benchmark suite. All four techniques do not differ much in the routed wirelength and the number of vias: the difference is below 0.2%. However, HPWL has the lowest runtime for placement, and therefore is an efficient estimation of the routed wire length.

	HPWL	**RSMT**	**RMST**	**RISA**
rWL	1.0000	0.9989	0.9988	1.0005
Vias	1.0000	0.9993	0.9984	1.0005
CPU Place	1.00	1.29	1.08	1.00
CPU Route	1.00	1.02	0.98	1.00

Table 6.3: Results of different techniques to estimate the routed wirelength (rWL) during placement. “CPU” is the runtime, either for placement, or for routing. The routing demand is estimated with RUDY, placement is done with Kraftwerk. Results are normalized to the results of HPWL. Based on all circuits of the IBM-PLACE 2.0 benchmark suite.

Two principle problems of estimating the routed wirelength should be pointed out. First, almost all estimation techniques represent each pin as a point in the x-y plane. In routing, each pin is represented by a rectangle, called “pin site”. This difference can result in that the estimation of the routed wirelength is higher than the exact routed wirelength. For example, imagine a two-pin net, where both pin sites almost touch each other. Hence, connecting both pins sites needs almost no wire. In contrast to this, connecting both pin points, which are located typically in the center of each pin sites, needs more wirelength. A second problem of estimating the routed wirelength is that routing considers the interaction (overlapping) between the nets. In contrast to this, traditional estimation techniques consider one net at a time, and ignore the interaction between the nets. This results in that the estimation of the routed wirelength is lower than the exact routed wirelength.

At last, the results presented in this chapter (see Table 6.2 and Table 6.3) can be summarized as follows: it is sufficient to use RUDY for estimating the routing demand, and it is sufficient to use the HPWL for estimating the routed wirelength. Other techniques may be a bit better, but consume much more runtime. Considering routability-driven placement, the placement with optimal routed wirelength is obtained in Kraftwerk by adjusting one parameter: w_{rout}.

Chapter 7

Legalization

Placement of circuits is done in two consecutive steps: global and final placement. In global placement, the modules are spread roughly on the chip, while considering different objectives like wirelength and routability. The previous chapters describe Kraftwerk, which is a fast global placer based on force-directed quadratic placement. Final placement itself consists also of two steps: legalization and detailed placement. Legalization means to remove the remaining module overlap of a global placement, and to align the modules to rows if necessary. Detailed placement is performed after legalization, and is the second step of final placement. In detailed placement, different objectives are further improved, for example total wirelength, or more complex objectives like design for yield (DFY), or design for manufacturing (DFM).

This chapter presents novel approaches for legalization. Detailed placement is not addressed in the following. To preserve the global placement as far as possible, the common objective of legalization is to minimize the module movement. In the following, two legalization approaches based on minimizing the quadratic movement are presented. With the quadratic norm, the minimum is found quickly. The first legalization approach "Puzzle" deals with legalizing macros. "Abacus" is the second legalization approach, which focuses on legalization standard cells. The separation between macros and standard cells is necessary here, because standard cells must be aligned to rows, and macros not. Moreover, there are millions of standard cells in a modern circuit, while there are just a few (about hundreds) macros. Thus, legalizing macros can be done with exhaustive approaches. In contrast to this, legalizing standard cells must be done quickly, concerning the runtime per standard cell.

Table 7.1 summarizes some properties of one module i (macro or standard cell). Both legalization approaches (Puzzle and Abacus) refer to these properties. The properties are similar to those shown in Table 5.1, which is used in global placement. However, (x'_i, y'_i) is the position of module i in the global placement now, and (x_i, y_i) the position in the legal placement. In detail, there are two meanings of "position": for macros, it refers to the center of the macro, and for standard cells, it refers to the lower left corner of the cell. This difference is made because it simplifies the later given problem formulations. w_i and h_i presented in Table 7.1 are the dimensions (with and height) of module i. The weight e_i of a module is for example the area of the module, or the number of pins located at the module.

In the next sections, the term "movement" is used with several norms: (7.1) is the quadratic Euclidean movement, (7.3) is the Euclidean movement, (7.2) is the Manhattan movement.

Property	**Explanation**
(x'_i, y'_i)	Position in global placement
(x_i, y_i)	Position in legal placement
w_i, h_i	Width, height
e_i	Weight (e.g. number of pins)

Table 7.1: Properties of module i (macro or standard cell).

Moreover, these are total movements, i.e., they are the sum of the movements of all modules. In addition, the movement of each module is weighted by $e_{\mu,i}$. The proposed quadratic programs optimize (7.1), or (7.2) in combination with linearization weights. The quality of a legal placement is measured by (7.3). These differences in the norms are made, because the quality is best measured in the Euclidean norm (7.3), but both other norms are best to minimize with numerical optimization.

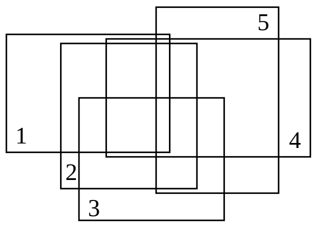
$$\mu_{Quad,Euclid} = \sum_{i=1}^{N} e_{\mu,i} \left([x_i - x'_i]^2 + [y_i - y'_i]^2\right) \tag{7.1}$$

$$\mu_{Manhattan} = \sum_{i=1}^{N} e_{\mu,i} \left(|x_i - x'_i| + |y_i - y'_i|\right) \tag{7.2}$$

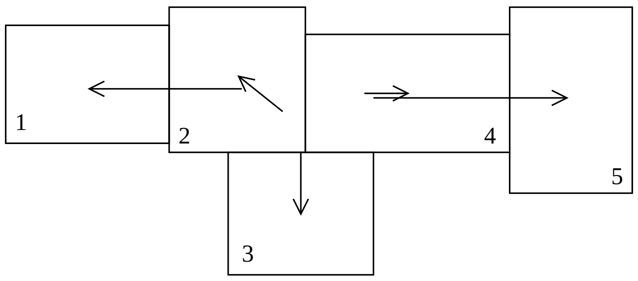
$$\mu_{Euclid} = \sum_{i=1}^{N} e_{\mu,i} \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2} \tag{7.3}$$

7.1 Puzzle: Macro Legalization

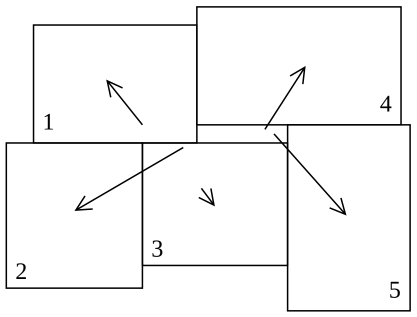
This section presents “Puzzle”, which is a fast legalization approach for macros. Macros are modules with various dimensions. Figure 7.1 (a) displays a global placement of macros. Two legal placements are shown in Figure 7.1 (b) and (c).



(a) Global Placement



(b) Legal Placement (constraint direction based on placement)



(c) Legal Placement (with constraint direction optimization)

Figure 7.1: Global placement of macros (a), and two legal placements of macros (b) and (c). The movement of each macro is displayed by an arrow. The start of the arrow reflects the position in the global placement, and the end of the arrow reflects the position in the legal placement. There, position refers to the position of the center of the macro. The total movement in (c) is about 25% lower than in (b).

The legalization of macros can be formulated by the following quadratic program (QP):

$$\text{QP:} \quad \begin{aligned} &\min \sum_{i=1}^{N} e_i \left(w_{x,i} \left[x_i - x_i' \right]^2 + w_{y,i} \left[y_i - y_i' \right]^2 \right) && (7.4) \\ &\text{s.t.} \quad \mathbf{A}\mathbf{p} \geq \mathbf{b} && (7.5) \end{aligned}$$

The objective (7.4) is similar to (7.1), and represents the sum of the weighted quadratic Euclidean movements of all N macros. $w_{x,i}$ and $w_{y,i}$ are used to linearize the quadratic movement. Since all weights (e_i, $w_{x,i}$, and $w_{x,i}$) are positive, the objective is convex. The constraint (7.5) assures that there is no overlap between the macros. The vector $\mathbf{p}$ reflects the legal positions of all macros, separated in x and y-direction:

$$\mathbf{p} = (x_1, x_2, ..., x_n, y_1, y_2, ..., y_N)^T \tag{7.6}$$

In general, two macros i and j do not overlap, if either the distance in x-direction, or the distance in y-direction is large enough. This constraint is described by:

$$|x_i - x_j| \geq \frac{1}{2}(w_i + w_j) \quad \vee \quad |y_i - y_j| \geq \frac{1}{2}(h_i + h_j) \tag{7.7}$$

In the rest of this section about Puzzle, the following is assumed for two indeces i and j: if the macros i and j are located at (x'_i, y'_i) and (x'_j, y'_j) in the global placement, then i and j are chosen such that $x'_i \geq x'_j$ and $y'_i \geq y'_j$. Based on this, (7.7) is transformed to:

$$x_i - x_j \geq \frac{1}{2}(w_i + w_j) \quad \vee \quad y_i - y_j \geq \frac{1}{2}(h_i + h_j) \tag{7.8}$$

With this, the relative order of the macros is preserved. In other words, if macro i is right of (above of) macro j in the global placement, then the ordering between both macros is the same in the legal placement. The constraints shown in (7.8) can be described by the matrix-vector notation $\mathbf{Ap} \geq \mathbf{b}$ for all macros. However, there is a "$\vee$" between the x- and y-constraint, which means that two macros must be overlap free *either* in x *or* in y-direction. In other words, a decision on the constraint direction (x or y) must be made here, and the decision influences highly the movement of the macros during legalization. In Figure 7.1 (a), the decision is done based on the global placement. In Figure 7.1 (b), the initial decisions are refined (optimized) by Tabu Search [GL97]. Comparing 7.1 (a) with 7.1 (b) demonstrates that this "constraint direction optimization" results in a lower total movement of the macros. In the following, some general aspects are described first, and then details of constraint direction optimization are presented.

7.1.1 Construction of Matrix $\mathbf{A}$ and Vector $\mathbf{b}$

The matrix-vector notation $\mathbf{Ap} \geq \mathbf{b}$, used in (7.5), represents in each row one constraint (7.8). In the following, the construction of matrix $\mathbf{A}$ and vector $\mathbf{b}$ is described. Matrix $\mathbf{A}$ has entry a_{kl} in row k and column l. Vector $\mathbf{b}$ has entry b_k in row k. First, $\mathbf{A}$ and $\mathbf{b}$ are initialized with zeros, and an index variable k is initialized with 1. Then, all pairs of macros are considered. One pair (i, j) contributes to $\mathbf{A}$ and $\mathbf{b}$ as follows:

1. If the constraint direction is x, then $\mathbf{A}$ and $\mathbf{b}$ are updated by:
   $a_{ki} \leftarrow a_{ki} + 1$, $a_{kj} \leftarrow a_{kj} - 1$, and $b_k \leftarrow b_k + \frac{1}{2}(w_i + w_j)$.
2. If the constraint direction is y, then $\mathbf{A}$ and $\mathbf{b}$ are updated by:
   $a_{k\,i+N} \leftarrow a_{k\,i+N} + 1$, $a_{k\,j+N} \leftarrow a_{k\,j+N} - 1$, and $b_k \leftarrow b_k + \frac{1}{2}(h_i + h_j)$.
3. The index variable k is increased by one:
   $k \leftarrow k + 1$.

7.1.2 Initial Legalization

Algorithm 2 shows how the initial legal placement is obtained. First, the macros are placed to the positions (x'_i, y'_i) of the global placement (line 1). Then, some iterations are done (line 2-8). In each iteration, the direction (x or y) of the constraints is determined based on the placement (line 3). Next section describes this important step of Puzzle. With the determination of the constraint direction, the matrix-vector notation of the overlap-free constraint $\mathbf{Ap} \geq \mathbf{b}$ is given. Then, the quadratic program (7.4) s.t. (7.5) is solved using some linearization iterations (line 4-7). With the linearization, the Manhattan movement (7.2) is optimized. At the end, a new placement with no overlap is obtained. Based on experimental results, about 3 linearization iterations are enough.

Since the constraint direction is decided based on a placement, and the constraint direction gives a new placement (via the quadratic program), both steps (determination of constraint direction and solving the quadratic program) are executed consecutively for some iterations. This is done in the “for” loop in line 2-8 in Algorithm 2. The loop is done until convergence, which means, the loop is executed until the quadratic program does not change the positions of the macros anymore. Based on experimental results, about 5 cycles for the loop are enough.

Algorithm 2: Initial macro legalization

```
1 Initialize $(x_i, y_i) \leftarrow (x'_i, y'_i)$;
2 for some iterations do
3     Create constraints based on placement $(x_i, y_i) \Rightarrow$ Matrix $\mathbf{A}$ and vector $\mathbf{b}$;
4     for some linearization iterations do
5         $w_{x,i} \leftarrow 1/|x_i - x'_i|, w_{y,i} \leftarrow 1/|y_i - y'_i|$;
6         Solve QP (7.4) s.t. (7.5) $\Rightarrow$ new positions $(x_i, y_i)$;
7     end
8 end
```

7.1.3 Constraint Direction based on Placement

One important step of Puzzle is to determine the direction (x or y) of each constraint. In the following, it is described how the direction of the constraint between two macros i and j is decided based on a placement. There, the decision is driven by moving both macros as little as possible. The two macros are located at (x_i, y_i) and (x_j, y_j). Two properties $\delta_{x,ij}$ and $\delta_{y,ij}$ can be computed:

$$\delta_{x,ij} = x_i - x_j - \frac{1}{2}(w_i + w_j) \tag{7.9}$$

$$\delta_{y,ij} = y_i - y_j - \frac{1}{2}(h_i + h_j) \tag{7.10}$$

These properties reflect the distance between both macros, and the dimensions of both macros. In detail, if $\delta_{x,ij} < 0$, the macros are overlapping in x-direction; if $\delta_{y,ij} < 0$, they are overlapping in y-direction. In both cases, $-\delta_{x,ij}$ and $-\delta_{y,ij}$ is the amount of overlap in each direction. Consequently, if $\delta_{x,ij} > \delta_{y,ij}$, then the movement for both macros to an overlap-free placement is lower in x-direction, than in y-direction. This exactly gives the decision of the constraint direction: if $\delta_{x,ij} > \delta_{y,ij}$, then the constraint direction is x, otherwise it is y. This decision is not made for overlapping macros only, but for all pairs of macros. In other words, there will be constraints between all pairs of macros. This is necessary for the convergence of Algorithm 2. Otherwise, macros, which were made overlap free in one iteration (see line 2-8 in Algorithm 2), have no constraint in the next iteration. Consequently, they will collapse back and will overlap again, which is not wanted.

Determining the constraint direction as described above minimizes just the movement between two macros. It does not minimize the total movement of all macros. Hence, the initial constraint direction can be good, but need not to be optimal.

7.1.4 Optimization of Constraint Direction

This Section describes the complete approach called "Puzzle", which is a novel method for macro legalization. The total movement is minimized by quadratic programming. The initial constraint directions are determined by the placement. Tabu Search is used to optimize the constraint directions, in order to minimize the total movement.

Before presenting Puzzle, some aspects are to be noted first. Algorithm 2, describing the determination of the initial legal placement, converges such that the QP solved in line 4 does not change the placement anymore. In such a placement, there are "essential" constraints, where two macros are abutting, i.e., there is no free space between the macros. These essential constraints have "$=$" instead of "$\geq$" in (7.8). All other constraints with "$>$" are not active. Hence, the legal placement, obtained by Algorithm 2, is characterized by essential constraints. The set of essential constraints, in combination with their directions, is called "configuration", and describes a legal placement; Tabu Search acts on these configurations. In the following, the terms "configuration" and "legal placement" are used interchangeably.

Algorithm 3 describes Puzzle, and the application of Tabu Search. The algorithm starts with an initial configuration (line 1), and optimizes iteratively the configuration (line 4-28). In each iteration, the neighboring configurations of the current configuration are evaluated (line 9-17). Each neighboring configuration is created by changing the direction of one essential constraint (line 10). Neighboring configurations, which are in the tabu-list, are ignored (line 18). Two special neighboring configurations are saved: the one with the best cost (line 20), and the one with the worst cost (line 21). After evaluating all neighboring configurations, the current configuration is compared with both saved neighboring configurations. If the best neighboring configuration has a better cost than the current configuration, then this neighboring configuration is used as the new current configuration (line 25). With this approach, Tabu Search is a greedy optimization method. However, if all neighboring configurations have a worse cost than the current configuration, then the worst configuration is used as the new current configuration (line 26). With this technique, Tabu Search has a "hill-climbing" ability, and local minima can be escaped. At the end of each optimization iteration, the new current configuration is appended in the tabu-list (line 27). With this method, configurations are only visited once.

The optimization iterations are done until a stopping criterion is triggered (line 28). Suitable stopping criterions are for example a maximal number of iterations, or a maximal increase in the cost, i.e., a maximal difference in the best cost so far and the current cost. At the end of Tabu Search, the best configuration is found in the tabu-list as the configuration with the lowest cost (line 29). This best configuration represents the best legal placement with a minimal total movement (line 30).

Algorithm 3: Puzzle: macro legalization with constraint direction optimization.

```
1  Do initial macro legalization (see Algorithm 2);
2  Initialize tabu-list;
3  Determine cost $c_{cur}$;
4  repeat
5      Save positions: $(\hat{x}_i, \hat{y}_i) \leftarrow (x_i, y_i)$;
6      $c_{best} \leftarrow \infty$;
7      $c_{worst} \leftarrow -\infty$;
8      foreach essential constraint do
9          Change direction;
10         Restore positions $(x_i, y_i) \leftarrow (\hat{x}_i, \hat{y}_i)$;
11         for some placement iterations do
12             Create constraints based on placement $(x_i, y_i)$, consider direction of the
               changed essential constraint $\Rightarrow$ Matrix $\mathbf{A}$ and vector $\mathbf{b}$;
13             for some linearization iterations do
14                 $w_{x,i} \leftarrow 1/|x_i - x'_i|$, $w_{y,i} \leftarrow 1/|y_i - y'_i|$;
15                 Solve QP (7.4) s.t. (7.5) $\Rightarrow$ new positions $(x_i, y_i)$;
16             end
17         end
18         if new configuration is not in tabu-list then
19             Determine cost $c$;
20             if $c < c_{best}$ then Save this configuration and positions as best, $c_{best} \leftarrow c$ ;
21             if $c > c_{worst}$ then Save this configuration and positions as worst, $c_{worst} \leftarrow c$;
22         end
23         Change direction;
24     end
25     if $c_{best} < c_{cur}$ then Restore best positions and configuration, $c_{cur} \leftarrow c_{best}$;
26     else Restore worst positions and configuration, $c_{cur} \leftarrow c_{worst}$;
27     Append cost $c_{cur}$, configuration, and positions in tabu-list;
28 until stopping criterion triggered ;
29 Scan tabu-list for best cost$\Rightarrow$Best positions;
30 Put macros to best positions;
```

Two details of Tabu Search about the constraint direction optimization are left to cover. First, the determination of the cost of one configuration, i.e., of one legal placement. A suitable cost is the total weighted Euclidean movement of all macros between the global placement and the legal placement, as described by (7.3) with $e_{\mu,i} = e_i$. The second and more interesting detail is how a neighboring configuration is created. This is done in line 9-17 of Algorithm 3. Starting from the current configuration, the direction of one essential constraint is changed (line 9). All macros are put back to the positions of the current configuration (line 10). In line 11-17, the neighboring configuration (i.e., the neighboring legal placement) is determined similar to the initial legal placement, as described in Algorithm 2. However, the constraint directions are created based on the placement now, *and* considering the direction of the changed essential constraint (line 12). If the changed essential constraint is between the macros i and j, then the constraint direction is not chosen based on $\delta_{x,ij}$ (7.9) and $\delta_{y,ij}$ (7.9) (see Section 7.1.3). Rather, the direction is the same as the direction of the changed essential constraint. The constraint direction between all other macros is chosen based on $\delta_{x,ij}$ and $\delta_{y,ij}$.

Figure 7.2 demonstrates the hill-climbing abilities of Tabu Search. Here, the cost of the current configuration in each optimization iteration is displayed. The cost represents the total movement of all macros between the global placement and the legal placement, as formulated in (7.3). Each configuration is a legal placement. In Figure 7.2, the cost of the initial configuration is rather high. Then, Tabu Search starts to change the constraint directions. Hence, the cost sinks over two optimization iterations. In iteration three, the cost increases. Hence, a "hill" in the cost function is climbed. After the hill, the cost in iteration four is lower. At the end, the cost increases, and the Tabu Search is stopped. Iteration four represents the best legal placement.

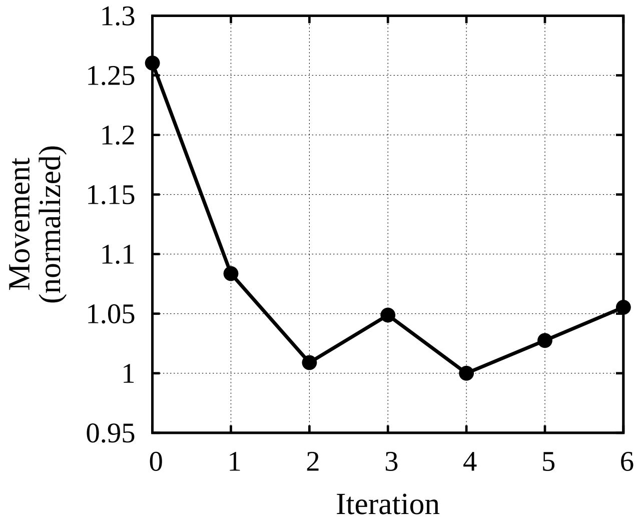


Figure 7.2: Minimization of the movement by optimizing the constraint directions with Tabu-Search.

7.1.5 Comparison

Using the linearization weights $w_{x,i}$ and $w_{y,i}$ in combination with some linearization iterations (see Algorithm 2 and 3), the quadratic program (7.4) s.t. (7.5) minimizes the total linear movement, i.e., the Manhattan movement (7.2). The quadratic program can be solved for example with OOQP [GW03]. Instead of using the quadratic program and some linearization iterations, a similar result (placement) is obtained by the following linear program (LP):

$$\text{LP:} \quad \begin{aligned} &\min \sum_{i=1}^{N} e_i \left(|x_i - x_i'| + |y_i - y_i'| \right) && (7.11) \\ &\quad \text{s.t.} \quad \mathbf{Ap} \geq \mathbf{b} && (7.12) \end{aligned}$$

Here, only the objective (7.11) changed. The constraint (7.12) is similar to (7.5). The linear program can be solved for example with GLPK [GGL]. This section presents some experimental results demonstrating that using the quadratic program with linearization iteration gives similar placements, but in lower runtime than using a linear program. The problem with linear programming is that the absolute movement $|x_i - x'_i|$ can not be minimized directly. Moreover, two auxiliary variables $\overline{x}_i$, $\underline{x}_i$, and four additional constraints are necessary:

$$\min |x_i - x'_i| \Rightarrow \min \overline{x}_i - \underline{x}_i \quad \text{s.t.} \quad \underline{x}_i \leq x_i \quad \underline{x}_i \leq x'_i \quad \overline{x}_i \geq x_i \quad \overline{x}_i \geq x'_i \tag{7.13}$$

This increases the numbers of variables and constraints in the LP compared to the QP. Thus, the LP needs more runtime to solve the same problem. Figure 7.3 displays the complexity of both approaches. Here, a global placement of a circuit with up to thousand macros is legalized. Different numbers N of macros are selected to be legalized, and all the selected macros are overlapping each other in the global placement. Except some minor glitches for small N, the quadratic program is always faster (lower runtime) than the linear program. Moreover, the computational complexity of the quadratic program is better than those of the linear program. Considering the quadratic program, the average-case computational complexity is $\Theta(N^2)$ for the initial placement. Applying the Tabu Search for optimizing the constraint directions, the complexity is $\Theta(N^{2.88})$. Using linear programming, the complexity is $\Theta(N^{2.55})$, and $\Theta(N^{3.71})$, respectively.

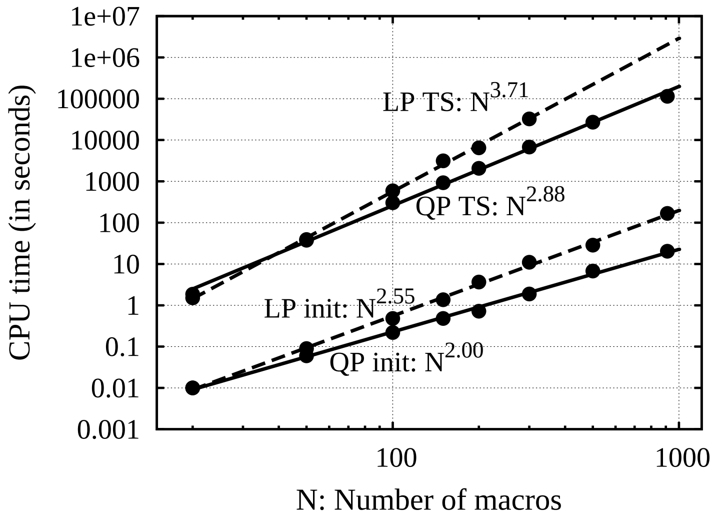


Figure 7.3: Runtime versus number of macros. QP: quadratic program, LP: linear program, init: initial placement, TS: applying Tabu Search for constraint direction optimization. Based on one global placement of a circuit with up to 911 macros.

Table 7.2 summarizes detailed results of Puzzle using quadratic programming and using linear programming. The results are based on the same global placement and the same circuit as used in Figure 7.3, which describes the computational complexity. In the following, the term "movement" means the total weighted Euclidean movement of all macros between the global placement and the legal placement, as described by (7.3) with $e_{\mu,i} = e_i$. Using quadratic programming, and based on the initial legal placement, Puzzle improves with Tabu

Search the movement by about 30%. Moreover, the improvement does not decline with increasing numbers of macros N. Hence, the Tabu Search approach is successful. Replacing the quadratic program and the linearization iteration with the equivalent linear program gives about the same results in the movement. However, the runtime is about factor four higher then.

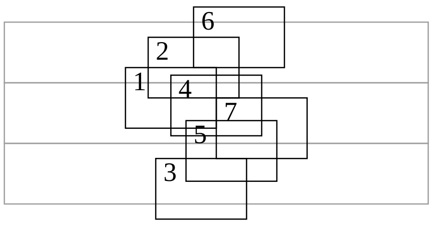
	Quadratic Program					Linear Program			
	Initial		Tabu Search			Initial		Tabu Search	
N # Macros	CPU [s]	Avg Mov	CPU [s]	Avg Mov	Impr [%]	CPU [s]	Avg Mov	CPU [s]	Avg Mov
20	0.01	25421	1.86	20446	19.57	0.01	25387	1.51	20664
50	0.06	39219	39.04	28207	28.08	0.09	39215	37.63	28811
100	0.22	57541	302.43	39030	32.17	0.48	58175	588.29	39914
150	0.48	64321	926.49	42352	34.16	1.36	64372	3142.14	41022
200	0.72	72653	2063.36	47585	34.50	3.65	73790	6479.32	49644
300	1.88	79369	6791.02	49145	38.08	11.00	83860	32497.50	55960
500	6.77	113492	27044.40	71326	37.15	28.52	120090	timeout	n.a.
911	20.29	238151	114673.00	173397	27.19	167.09	232337	timeout	n.a.
Average:	**1.00**	**1.00**	**1.00**	**1.00**		**3.86**	**1.01**	**2.51**	**1.03**

Table 7.2: Results of Puzzle (with quadratic programming) legalizing one global placement of a circuit with up to 911 macros. “Avg Mov” means the average Euclidean movement $||\mu||_2/N$ (7.3). “Impr” is the improvement in the movement between initial placement and after applying Tabu Search.

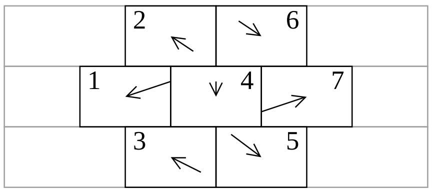
In summary, the results shown in Figure 7.3 and in Table 7.2 demonstrate that using quadratic programming in combination with some linearization iteration is better than using the equivalent linear program. Better means the runtime is lower and the total linear movement is equivalent. For circuits with some macros, Tabu Search can be used to improve the macro movement significantly. However, for circuits with hundred or more macros, Table 7.2 shows that Tabu search is not applicable due to the high runtime. To cope with this, two approaches can be used. First, not all macros are legalized with Puzzle, but only big macros. This decreases the number N of macros, for which Puzzle is applied to. The remaining macros can be legalized with Tetris [Hil02] (see Section 7.3). The second approach is to stop global placement when the module overlap is rather low (e.g., 5%), and not at 20% overlap. Based on such a global placement, the initial decisions for the constraint direction, which are made based on the global placement, are quite good. Tabu Search will not improve the movement much here. Thus, Tabu Search needs not be applied, and the initial legal placement is sufficient. However, with more iterations spent in the global placement, global placement takes more runtime. In addition, in mixed-size circuits, the standard cells are moved farther during global placement, resulting in a higher netlength. Hence, the second approach in applying global placement longer is only applicable for circuits, which consist only of macros.

7.2 Abacus: Standard Cell Legalization

Previous Section 7.1 described Puzzle, a novel approach based on quadratic programming and Tabu Search to legalize macros. This section describes Abacus, a new method based on quadratic programming and dynamic programming to legalize standard cells. In contrast to macros, standard cells all have the same height, and have to be aligned to the rows of the chip. In addition, in modern circuits, the number of standard cells is some decades greater than the number of macros. In Figure 7.4 (a), a global placement of standard cells is displayed. Figure 7.4 (b) shows the legal placement obtained by Abacus.



(a) Global Placement



(b) Legal Placement

Figure 7.4: Global and legal placement for standard cells. The movement of each cell is displayed by an arrow. The start of the arrow reflects the position in the global placement, and the end of the arrow reflects the position in the legal placement.

In case that the circuit has standard cells and macros, it is assumed that the macros are legalized first, for example by using Puzzle. Furthermore, rows, which are blocked (e.g., by macros) have to be sliced into new rows, such that all new rows are not blocked anymore.

Algorithm 4: Abacus: legalization of standard cells.

```
1  Sort cells according to x-position;
2  foreach cell $i$ do
3      $c_{best} \leftarrow \infty$;
4      foreach row $r$ do
5          Insert cell $i$ into row $r$;
6          PlaceRow $r$ (trial);
7          Determine cost $c$;
8          if $c < c_{best}$ then $c_{best} = c$, $r_{best} = r$;
9          Remove cell $i$ from row $r$;
10     end
11     Insert Cell $i$ to row $r_{best}$;
12     PlaceRow $r_{best}$ (final);
13 end
```

Algorithm 4 describes Abacus. First, the cells are sorted according to their x-position (line 1). Then, the cells are legalized one at a time (line 2-13). The legalization of one cell i is done by moving it over the rows (line 4-10). In each row, the cell is inserted according to its x-position in the global placement (line 5). Then, “PlaceRow” (line 6) places all cells of the row such that their total movement is minimal and they are not overlapping. PlaceRow is described in the next section. After PlaceRow is called, the cost of the new position of cell i is determined (line 7), e.g., by the movement of cell i between its position in the global placement and its new position in the current row. At last, the cell i is removed from the current row (line 9). After the cell i is moved over the rows, it is inserted into the best row (line 11). The best row is the row with the lowest cost (line 8). During the movement of the cell i over the rows (line 4-10), i.e., during the trial mode, the results of PlaceRow are treated as temporary positions, which means that the cells are not really moved to these positions. Hence, the best row needs to be placed again (line 12), and the results of PlaceRow are treated as the final legal positions. This means, the cells are actually placed to these positions. Since one cell at a time is legalized, and the cell is placed to the best row, Abacus is a greedy algorithm. However, already legalized cells within the rows are moved (by PlaceRow), which improves the total movement.

Different issues should be noted here. First, the sorting of the cells according to their x-position can be done either in increasing order or in decreasing order. Both directions should be tested because the results of each direction can be different and the best result should be used. Experiments showed that the difference in the total movement between both sort directions is about 0.5%. Another issue is that each cell need not be moved over all rows of the chip (line 4-10). Rather, each cell is first moved to the nearest row (according to the global position) and then moved above and below this row. For each row, a lower bound of the cost is computed by assuming that the cell is only moved vertically. If the lower bound exceeds the minimal cost of an already found legal position, then the movement of the cell over the rows can be stopped. This method limits the movement only to some rows and improves the CPU time of legalization drastically. At last, it should be noted that the cells are inserted into the rows in order of their x-position in the global placement. Since the cells are processed according to their global x-position (line 1,2), inserting a cell into a row means either to append the cell as the last cell in the row (if sorted in increasing order), or as the first cell (if sorted in decreasing order).

7.2.1 PlaceRow

The core of Abacus is to optimize the total quadratic movement of all cells within one row. This optimization is called PlaceRow, and it is used several times for each cell during legalization (see Algorithm 4). In the following, PlaceRow is described.

In PlaceRow, it is assumed that the row has N_r standard cells, indexed from 1 to N_r. Table 7.1 shows the different properties of one cell i. Given is the position (of the lower left corner of the cell) in the global placement (x'_i, y'_i), the width w_i, and the weight e_i. The weight can be for example one, the area of the cell, or the number of pins of the cell. The cells in the row are sorted according to their global x-position, i.e., $x'_i \geq x'_{i-1}$. PlaceRow determines the legal x-position x_i of each cell. The legal y-position y_i is obtained beyond PlaceRow by moving the cell over the rows (see line 4-10 in Algorithm 4). Based on these definitions, PlaceRow is described by the following quadratic program:

$$\text{QP:} \quad \begin{aligned} &\min \sum_{i=1}^{N_r} e_i \, [x_i - x_i']^2 && (7.14) \\ \text{s.t.} \quad & x_i - x_{i-1} \geq w_{i-1} \quad i = 2, ..., N_r && (7.15) \end{aligned}$$

The objective (7.14) describes the total, weighted, and squared movement of all cells between the global position x_i' and the legal position x_i. Moreover, the objective is convex, since all weights e_i are positive. Furthermore, the objective is similar to (7.1) with $N = N_r$, $e_{\mu,i} = e_i$, and since all cells in the row have the same y-position, $y_i = y_i'$. The constraint (7.15) assures that there is no overlap between the cells. In addition, the constraint preserves the relative order of the cells, i.e., cell a is placed left of cell b if a is left of b in the global placement.

The quadratic program of PlaceRow (7.14) s.t. (7.15) is similar to the quadratic program of Puzzle (7.4) s.t. (7.5). However, PlaceRow does not utilize linearization weights, because it is called several times for each cell, and using linearization weights would mean to increase the number of calls. In addition, PlaceRow does not optimize the constraint direction, but all constraints between the cells in one row have to be in x-direction.

Similar to the quadratic program of Puzzle, the quadratic program of PlaceRow (7.14) s.t. (7.15) can be solved with OOQP [GW03] for example. However, solving quadratic programs with "$\geq$" constraints is time consuming in general. If the same solution of the quadratic program (i.e., the same legal placement) is found by equality constraints, then the quadratic program is solved quite fast by solving one linear equation. The situation that equality constraints are sufficient is given if all cells of one row are abutting in the legal placement. There, two cells are "abutting" if there is no free space between them in the legal placement. With only equality constraints, (7.15) is transformed to:

$$x_i = x_1 + \sum_{k=1}^{i-1} w_k \quad i = 2, ..., N_r \tag{7.16}$$

Inserting (7.16) in the objective of (7.14) gives a quadratic function, only depending on x_1. The minimum of this function is obtained by setting its derivative with respect to x_1 to zero, which gives:

$$\underbrace{\sum_{i=1}^{N_r} e_i}_{\hat{e}} \; x_1 - \underbrace{\left[e_1 x_1' + \sum_{i=2}^{N_r} e_i \left[x_i' - \sum_{k=1}^{i-1} w_k \right] \right]}_{\hat{q}} = 0 \tag{7.17}$$

Table 7.3 shows the iterative calculation of $\hat{e}$, $\hat{w}$, and $\hat{q}$, which depends only on given properties of the cells: x_i', w_i, and e_i. Executing the iterations up to $i = N_r$, $\hat{e}$ is the total weight of all N_r cells, and $\hat{w}$ is the total width of all N_r cells. $\hat{q}$ is used in (7.17), and gives the optimal position x_1 of cell $i = 1$:

$$\hat{e}\, x_1 - \hat{q} = 0 \quad \Leftrightarrow \quad x_1 = \frac{\hat{q}}{\hat{e}} \tag{7.18}$$

Using (7.16), the optimal positions x_i of the remaining cells ($i = 2, ..., N_r$) in the row are determined. At this point, the quadratic program, and thus PlaceRow, is solved based on one linear equation (7.18) — assuming equality constraints.

Init	**Iteration ($i = 1, 2, ..., N_r$)**
$\hat{e} = 0$	$\hat{e} \leftarrow \hat{e} + e_i$
$\hat{q} = 0$	$\hat{q} \leftarrow \hat{q} + e_i \left[x'_i - \hat{w} \right]$
$\hat{w} = 0$	$\hat{w} \leftarrow \hat{c} + w_i$

Table 7.3: Iterative calculation.

7.2.2 Implementation by Dynamic Programming

However, the equality constraints, which are used to obtain (7.18), are just allowed for abutting cells, and not in general for all cells in one row. Therefore, a method is necessary to detect clusters of cells, where all cells in the clusters are abutting, and the clusters themselves do not abut. The optimal position of a cluster is found then by solving (7.18) for this cluster. Here it should be noted that (7.18) is obtained by assuming that all cells $i = 1, ..., N_r$ in the row are in one cluster. The clustering method, and solving PlaceRow by dynamic programming are shown in this section. The properties of one cluster c are summarized in Table 7.4. $n_{\mathit{first},c}$ is the first cell in the cluster c, and $n_{\mathit{last},c}$ is the last cell in the cluster c. $N_{\mathit{clus},c}$ is the number of cells in cluster c. $x_{\mathit{clus},c}$ is the left x-position of cluster c, $e_{\mathit{clus},c}$ represents the total weight of the cells in the cluster c, and $w_{\mathit{clus},c}$ is the total width of the cluster c.

Property	**Explanation**
$n_{\mathit{first},c}$	First cell of the cluster
$n_{\mathit{last},c}$	Last cell of the cluster
$N_{\mathit{clus},c}$	Number of cells in the cluster, $N_{\mathit{clus},c} = n_{\mathit{last},c} - n_{\mathit{first},c} + 1$
$x_{\mathit{clus},c}$	Optimal position (lower left corner)
$e_{\mathit{clus},c}, w_{\mathit{clus},c}, q_{\mathit{clus},c}$	Values similar to Table 7.3.
$e_{\mathit{clus},c}$	Total weight
$w_{\mathit{clus},c}$	Total width
$q_{\mathit{clus},c}/e_{\mathit{clus},c}$	Optimal position

Table 7.4: Properties of cluster c.

Algorithm 5 shows the implementation of PlaceRow by dynamic programming. The algorithm starts in line 1-13 with iteratively clustering the cells, and determining the optimal position of each cluster. Here, the cells $i = 1, ..., N_r$ are processed in increasing order (line 1) according to their global x-position x'_i, i.e., $x'_i \geq x'_{i-1}$. In other words, the cells are processed from "left" to "right". If cell i is the first cell, or if it does not overlap with the last cluster (line 3), then a new cluster is created containing the cell i (line 4-8). Otherwise, the cell i is added to the last cluster (line 10), and the last cluster is recursively collapsed with its predecessor cluster (the next left cluster) as long as the clusters are overlapping (line 11, and line 27-36, respectively). During the clustering, the iterative calculation of $e_{\mathit{clus},c}$, $w_{\mathit{clus},c}$, and $q_{\mathit{clus},c}$ is done in line 24-26, which is similar to Table 7.3. The optimal position $x_{\mathit{clus},c}$ of cluster c is deter-

Algorithm 5: PlaceRow: places all cells in one row optimally, i.e., with minimal total movement. Solves (7.14) s.t. (7.15).

```
      // Determine clusters and their optimal positions $x_{clus,c}$:
 1    for $i = 1, ..., N_r$ do
 2        $c \leftarrow$ Last cluster;
          // First cell or cell $i$ does not overlap with last cluster:
 3        if $i = 1$ or $x_{clus,c} + w_{clus,c} \leq x'_i$ then
 4            Create new cluster $c$;
 5            Init $e_{clus,c}, w_{clus,c}, q_{clus,c}$ to zero;
 6            $x_{clus,c} \leftarrow x'_i$;
 7            $n_{first,c} \leftarrow i$;
 8            AddCell($c, i$);
 9        else
10            AddCell($c, i$);
11            Collapse($c$);
12        end
13    end
      // Transform cluster positions $x_{clus,c}$ to cell positions $x_i$:
14    $i \leftarrow 1$;
15    for all clusters $c$ do
16        $x = x_c(c)$;
17        for $i \leq n_{last,c}$ do
18            $x_i \leftarrow x$;
19            $x \leftarrow x + w_i$;
20        end
21    end
22    Function AddCell($c, i$):
23    $n_{last,c} \leftarrow i$;
24    $e_{clus,c} \leftarrow e_{clus,c} + e_i$;
25    $q_{clus,c} \leftarrow q_{clus,c} + e_i \cdot (x'_i - w_{clus,c})$;
26    $w_{clus,c} \leftarrow w_{clus,c} + w_i$;
27    Function Collapse($c$):
      // Place cluster $c$:
28    $x_{clus,c} \leftarrow q_{clus,c}/e_{clus,c}$;
      // Limit position between $x_{min}$ and $x_{max} - w_{clus,c}$
29    if $x_{clus,c} < x_{min}$ then $x_{clus,c} = x_{min}$;
30    if $x_{clus,c} > x_{max} - w_{clus,c}$ then $x_{clus,c} = x_{max} - w_{clus,c}$;
      // Overlap between $c$ and its predecessor $c'$?:
31    $c' \leftarrow$ Predecessor of $c$;
32    if $c'$ exists and $x_{clus,c'} + w_{clus,c'} > x_{clus,c}$ then
          // Merge cluster $c$ to $c'$:
33        for $i = n_{first,c}$ to $n_{last,c}$ do AddCell($c', i$);
34        Remove cluster $c$;
35        Collapse($c'$);
36    end
```

mined in line 28. This is similar to (7.18). In line 29 and 30, the position of a cluster is limited such that the left corner $x_{clus,c}$ is right of the starting position x_{min} of the row, and the right corner $x_{clus,c} + w_{clus,c}$ is left of the ending position x_{max} of the row. In line 14-21 of Algorithm

5, the optimal positions x_i of all cells are determined based on the optimal positions $x_{clus,c}$ of the clusters to which the cells belong. After that, PlaceRow, and the quadratic program (7.14) s.t. (7.15) are solved.

The described dynamic programming approach for PlaceRow is optimal in the result, because the clusters, which are formed during the algorithm, are placed to their optimal positions (see line 28). Moreover, each cluster consists only of abutting cells, because a cell (or a cluster) is clustered with its left neighbor, only if they are overlapping (line 32-35). As a consequence, the clusters themselves do not abut.

7.2.3 Worst-Case Computational Complexity

The worst-case complexity of PlaceRow is given by the number of calls to function “AddCell” (line 22-26). AddCell is called once for each cell (line 8 and 10). During recursive collapsing, AddCell is called overall at most $i - 1$ times for cell i (line 33). This extreme situation represents that all cells are in one cluster at the end. Thus, AddCell is called maximal i times for cell i. AddCell itself has constant runtime. With N_r the number of cells in one row, the worst-case complexity of PlaceRow is $\sum_{i=1}^{N_r} i = O(N_r^2)$. Another critical part for the complexity of PlaceRow is line 18-19. However, this part is executed only once per cell, which alone would give only $O(N_r)$ for PlaceRow.

Based on the complexity of $O(N_r^2)$ for PlaceRow, the worst-case computational complexity of Abacus (Algorithm 4) can be analyzed. With N the number of cells in the circuit, the “foreach loop” in line 4-10 of Algorithm 4 is called N times. With R the number of rows, one “foreach loop” has R cycles. In each cycle, PlaceRow is called. With at most $\hat{N}_r$ cells in one row, the complexity of PlaceRow is limited by $O(\hat{N}_r^2)$. Since all of this is executed in a nested way, the worst case complexity of Abacus is $O(N\, R\, \hat{N}_r^2)$.

To obtain a complexity of Abacus, which just depends on N, approximations for R and $\hat{N}$ are necessary. Assuming that the standard cells are quadratic (same width and height), and the chip area is also quadratic, the number of rows is $R \approx \sqrt{N}$. The upper bound for the number of cells in one row is the same $\hat{N}_r \approx \sqrt{N}$. This gives the complexity of Abacus by $O(N^{2.5})$.

7.2.4 Average-Case Computational Complexity

Figure 7.5 displays the runtimes of legalizing various circuits with Abacus. N represents the numbers of standard cells per circuit. The results with $N < 10^6$ are based on the IBM-PLACE 2.0 benchmark suite [YCS02], the other results are based on the ISPD 2005 contest benchmark suite [NAV$^+$05] and on the ISPD 2006 contest benchmark suite [DES]. With the almost linear average-case computational complexity of $\Theta(N^{1.19})$, Abacus can easily cope with future circuits having an increasing N. Moreover, the worst-case complexity of $O(N^{2.5})$ shown in the previous section is not reached by experiments.

7.2.5 Comparison

Tetris [Hil02] is similar to Abacus in that the cells are sorted according to their position first, and then legalized one at a time then. Next Section 7.3 presents Tetris. The main difference

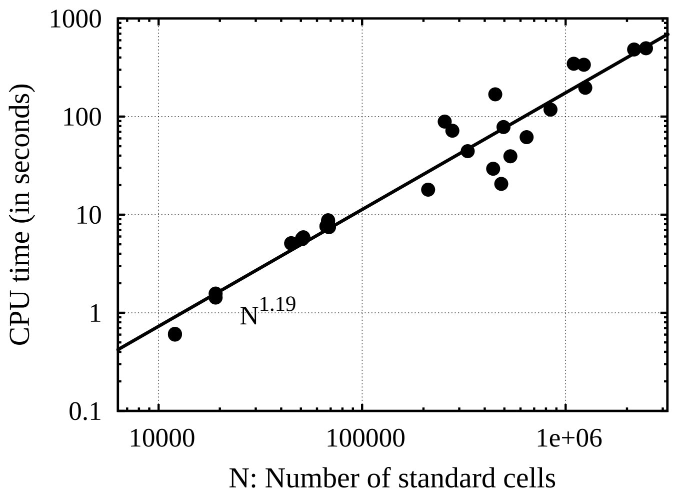


Figure 7.5: Average computational complexity of Abacus.

between Tetris and Abacus is that cells, which are legalized once, are not moved anymore in Tetris. In contrast to this, Abacus applies PlaceRow whenever a cell is moved to a row, and PlaceRow places all cells within a row such that there total quadratic movement is minimal. Consequently, Abacus moves already legalized cells during legalization. Therefore, the total movement of the cells during legalization is supposed to be lower in Abacus than in Tetris. Here, and in the following, movement is determined by (7.3) with $e_{\mu,i} = 1$, which means the movement is the (unweighted) Euclidean movement of the cells between global and legal placement. Figure 7.6 shows the histogram of the movement. The perfect histogram would be a peak with a relative frequency of one at a movement of zero, representing that all cells are not moved. However, the cells in the global placement are overlapping and are not aligned to the rows. Therefore, the cells are moved during legalization. Compared to Tetris, the histogram of the movement using Abacus is better, the cells are moved less and the peak is nearer to zero movement. The average movement is about 30% lower in Abacus than in Tetris.

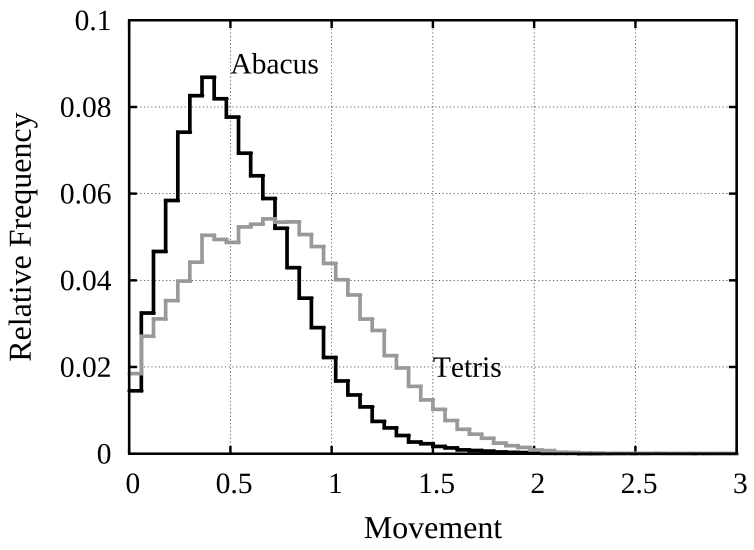


Figure 7.6: Movement histogram of Abacus and of Tetris. Based on ibm12e of the IBM-PLACE 2.0 benchmark suite. Movement is normalized to the average dimension of the cells.

7.3 Tetris

In this thesis, there are some references to the legalization approach Tetris [Hil02]. Therefore, this approach is presented shortly in the following. As Tetris can be used for macros and for standard cells, the term “module” is used below instead of “macro” or “cell”. Based on design rules, Tetris assumes that a grid structure exists, which gives a set of available x- and y-positions. For example, the available y-positions are given by rows, and the available x-positions are given by the minimum feature size of the technology, which is used to fabricate the circuit. Algorithm 6 describes Tetris. First, the modules are sorted according to their x positions (line 1). Then, the modules are legalized one at a time (line 2-13). The legalization of one module i is done by moving the module over the chip according to the available x and y positions (line 4 and 5). If module i fits at the current position (x, y), i.e., the module does not overlap with already legalized modules, then the cost of this position is determined (line 7). For example, the cost is the movement of module i between global placement and the current position (x, y). Or the cost is the length of the nets adjacent to module i. After the movement of module i over the chip, the module is placed to the best legal position (line 12). The best legal position is the one with the lowest cost (line 8).

One advantage of Tetris is the simple implementation. One feature of Tetris that can be viewed as an advantage or as an disadvantage, is that the relative order of the modules is not preserved. This means, if module a is left of (or above of) b in the legal placement, then module a could have been right of (or below of) b in the global placement. As a consequence, the legal placement obtained with Tetris can have a lower HPWL netlength than the legal placements obtained with the previous presented approaches Abacus and Puzzle. However, both latter approaches preserve the relative order of the modules, and thus preserve the global placement better than Tetris. The main disadvantage of Tetris is that once a module is legalized, it will not be moved anymore. Compared to Abacus, this results in a higher total

Algorithm 6: Tetris: greedy legalization.

```
Sort modules according to x-position;
foreach module $i$ do
    $c_{best} \leftarrow \infty$;
    foreach $x$ do
        foreach $y$ do
            if module $i$ fits at $(x, y)$ then
                Determine cost $c$;
                if $c < c_{best}$ then $c_{best} = c$, $x_{best} = x$, $y_{best} = y$;
            end
        end
    end
    Place module $i$ to $(x_{best}, y_{best})$;
end
```

movement of all modules during legalization (see Section 7.2.5). This, in combination with not preserving the relative order, results in that the global placement is not very well preserved in Tetris. Consequently, the legal placements obtained with Tetris have a higher routed wirelength than the legal placement obtained with Abacus (see Section 8.3).

Chapter 8

Experimental Results

This chapter demonstrates the high quality and extremely low runtime of the presented approaches for global placement, including routability-optimization, and for legalization. The results of various benchmark suites are shown. All results are legal placements, and all runtimes report the total runtime of the complete placement flow. To obtain the results, the following placement flow is used:

1. Global placements are obtained by “Kraftwerk”.
2. Nets are modeled in global placement by the “Bound2Bound” net model.
3. Routability is optimized during global placement by integrating the routing demand estimation approach “RUDY” in Kraftwerk.
4. Legalization of global placement is done depending on the circuit type and on the objective of placement:
   (a) Standard cells in routability-optimized placements are legalized with “Abacus”.
   (b) Big macros in mixed-size circuits are legalized with “Puzzle” using Tabu Search.
   (c) All macros in the floorplacement circuits are legalized with “Puzzle” without Tabu Search. There, floorplacement means there are about thousands of modules with various dimensions, the dimensions are all fixed, and the modules have to be placed overlap-free within a given placement region.
   (d) In benchmark suites, where the quality is measured in HPWL netlength and not in routed wirelength, standard cells are legalized with Tetris [Hil02]. This is done because Tetris can optimize the HPWL netlength during legalization. However, the movement of the standard cells is much higher then. Hence, Tetris is not used for legalization a routability-optimized global placement.
   (e) The remaining small macros in mixed-size circuits are legalized with Tetris. This is done because Puzzle in combination with Tabu Search would consume too much runtime. Tetris is much faster here, however, the movement of the macros increases.

5. Detailed placement is used to improve the netlength of the legal placement. A simple and greedy approach is used here: single modules are rotated, or pairs of neighboring modules are exchanged. In addition, the modules in each row are placed such their total HPWL netlength is minimal by using an approach similar to [KTZ99, BV00]. There, the alignment of the modules to the rows, and the ordering of the modules within the rows is not modified.

Since global placement is the first step in the placement flow, and determines mainly the result, the complete placement flow as presented above will be denoted as “Kraftwerk” in the rest of this chapter.

All benchmark suites are placed on an AMD Opteron 248 machine with 8 GB RAM running at 2.2 GHz. The memory usage of the biggest benchmark is below 4 GB. On average, about 80% of the total runtime is spent in global placement. The remaining 20% are spent in legalization and detailed placement. Moreover, most runtime of global placement is used to solve the systems of linear equations: (5.46) for x-direction, and a similar one for y-direction. Since both directions can be solved concurrently, the two CPU cores of the AMD Opteron could be used, which would give a speedup of almost two. However, to have comparable runtime, this was not done.

To compare the runtimes with other published runtimes, the runtimes are scaled according to the SPEC CPU2000 benchmark [Cor]. This scaling factor will be noted as “CPU scaling” in the following. All HPWL netlengths, and all routed wirelengths are expressed in meters. The runtimes are denoted by “CPU” and are in seconds.

In all benchmark suites, the chip area of the circuits, and the metrics to measure the quality of a placer are given. Mostly, the HPWL netlength or the routed wirelength are used as quality metrics. However, the ISPD 2006 contest benchmark suite [ISP06] uses various quality metrics, and the most important one considers routability and runtime. Most benchmark suites were introduced in publications. However, two of them, namely the ISPD 2005 and 2006 contest benchmark suites [ISP05, ISP06], were introduced in two international placement contests, and various academic teams attended these contests. The circuits of the contest benchmark suites were given by the IBM corporation and represent modern integrated circuits.

In the following, two key features of Kraftwerk are demonstrated first: stability and support of the engineering change order. After that, results of various benchmark suites are presented.

8.1 Stability

One important feature of Kraftwerk is the stability of the placement algorithm. A placement algorithm is stable, if for a small change in the input (i.e., in the circuit), the output (i.e., the placement) changes also just a bit [ANVY05]. Today, small changes in the circuit arise frequently during the design flow. After running the whole design process for the first time, important specifications like maximal clock frequency are evaluated based on the placed and routed circuit. Mostly, the specifications are not met, and the circuit is modified, for example by sizing some gates [BJ90], or by inserting buffers [vG90]. After these small changes in the circuit the design process is restarted and placement is performed again. These cycles in the design flow are executed until all specifications are met. To have convergence in the design flow, the placement algorithm must be stable and the changes in the placement must be low. Therefore, stability is as important for a placement algorithm as giving high quality placements [ANVY05].

In [ANVY05], different stability metrics are presented. Amongst others, the stability is measured by the change of the pin positions between two placements A and B. Placement A is obtained based on the original circuit, and placement B is obtained based on the gate-sized circuit, i.e., based on the changed circuit. Let (x_i^A, y_i^A) be the position of pin i in placement A, and (x_i^B, y_i^B) be the position of pin i in placement B. For each net $n \in \mathcal{N}$, the geometric center position is also given, (x_{cn}^A, x_{cn}^A) and (x_{cn}^B, y_{cn}^B), respectively. For one net n with P pins, indexed from 1 to P, the perturbation D_n is determined as follows [CS07]:

$$D_n = \sum_{i=1}^{P} \left| \left| x_i^A - x_{cn}^A \right| - \left| x_i^B - x_{cn}^B \right| \right| + \left| \left| y_i^A - y_{cn}^A \right| - \left| y_i^B - y_{cn}^B \right| \right| \tag{8.1}$$

The perturbation D_n is zero, if the pin positions are the same in A and B. D_n is also zero, if the relative position between the pins and the geometric centers of each net do not change, i.e., $\left| x_i^A - x_{cn}^A \right| = \left| x_i^B - x_{cn}^B \right|$. Hence, the lower D_n is, the smaller are the changes in the placements, and the more stable is the placement algorithm. Considering all nets of a circuit, the average of D_n can be considered, the root mean square (RMS) of D_n, or the maximum of D_n. In Table 8.1, these metrics for one test case, and using different placers are presented. The test case is based on the circuit bigblue1 of the ISPD 2005 contest benchmark suite. The circuit is changed by doubling the width of randomly chosen modules, either of 10% of all modules, or of 20% of all modules. The results of Morph and Capo are taken from [CS07]. The results in Table 8.1 demonstrate that Kraftwerk is stable, because the perturbation in D_n is very low compared to other placers. Both other placers (Morph and Capo) have a higher perturbation, which ranges between factor two higher, up to factor seven higher. Here, it should be noted that in particular, the placer Morph targets stability [CS07]. Moreover, other results than presented in the table below are not available in [CS07].

With the excellent stability of Kraftwerk, this placement approach is suitable to be used in the everyday design process, and supports best the convergence of the design process and achieving timing closure.

Change in	**Kraftwerk**				**Morph**				**Capo**			
input	HPWL	Avg	RMS	Max	HPWL	Avg	RMS	Max	HPWL	Avg	RMS	Max
10%	101.01	175	1047	413,610	106	634	3590	1,010,000	116	1190	9870	3,490,000
20%	104.02	183	1577	701,873	109	645	3740	1,180,000	120	1200	9080	2,880,000
Average	**1.00**	**1.00**	**1.00**	**1.00**	**1.05**	**3.57**	**2.90**	**2.06**	**1.14**	**6.68**	**7.59**	**6.27**

Table 8.1: Results representing stability. The values in the columns “Avg”, “RMS”, and “Max” represent the average, root mean square, and maximum in net perturbation D_n.

8.2 Engineering Change Order

The previous section describes that changes in the circuit are part of the everyday design flow. The changes arise, because a placed circuit does not meet all specifications like maximal clock frequency or area consumption. To meet the specification, the circuit is changed slightly, which is called Engineering Change Order (ECO) in this section. After ECO, the circuit needs to be placed again. To speed up the design process, placement is not executed from scratch, but from a previous given placement. This section evaluates the ECO feature of Kraftwerk. The experiments are based on the circuit bigblue1 of the ISPD 2005 contest benchmark suite. In the first run, the original circuit is placed, and global placements at different placement iterations are saved. In the second run, the circuit is modified and placed again, either from scratch, or starting with the saved global placements of the first run. The circuit is modified by randomly choosing 10% of all moduels, and by doubling the width of these modules. Table 8.2 displays the results of the second run. There, the placement quality, measured in HPWL netlength, does not change significantly. However, the runtime (CPU) is decreasing drastically. For example, the runtime is more than 80% lower, if the modified circuit is not placed from scratch, but from the last given global placement of the first run (given at iteration 25). Thereby, the placement quality changes only by about 0.5%. Therefore, Kraftwerk supports ECO best, mainly because of the hold force, which decouples each placement iteration from the previous one. Consequently, the placement process can be restarted easily at any placement iteration.

Mode	**HPWL**	**CPU**
From scratch	101.01	435
With iteration 5	0.17%	-40%
With iteration 10	0.28%	-51%
With iteration 15	0.26%	-69%
With iteration 20	0.28%	-78%
With iteration 25	0.49%	-82%

Table 8.2: ECO feature of Kraftwerk. After gate sizing a circuit, the placement process is restarted, either from scratch, or with a placement of the previous placement run.

8.3 IBM-PLACE 2.0 Benchmark Suite

The IBM-PLACE 2.0 benchmark suite [YCS02] consists of sixteen circuits (ibm03e/h-ibm06e/h do not exist) with up to 68k modules and 68k nets. The quality of placement is measured in the routed wirelength and in the number of vias. Hence, this is a routability-driven benchmark suite. The routing is done with Cadence WarpRoute 2.3.33, and includes final routing.

Table 8.3 shows the results of Kraftwerk and of other state-of-the-art placement approaches. The results of ROOSTER, mPL, and APlace are taken from [RM07], using a CPU scaling of 0.91. Compared to other placement approaches, Kraftwerk offers results with the lowest routed wirelength and the lowest number of vias. The difference to the other placement approaches ranges from 0.4% to 11%. In addition, Kraftwerk is 14 times faster for placement than ROOSTER. Runtimes of other placers are not available. Moreover, the placements of Kraftwerk are routed in the lowest runtime. The routing of other placements needs between 40% and 300% more runtime. In addition, all placements of Kraftwerk are routable, i.e., there are no routing violations. In summary, the results of Table 8.3 demonstrate the efficiency of Kraftwerk using RUDY for estimating the routing demand during global placement, and using Abacus for legalization.

In Table 8.4, a comparison between Abacus and Tetris for legalization is given. The remaining placement flow of Kraftwerk is not changed. The results shown in the columns "Abacus" are the same as the result shown in the columns "Kraftwerk" in Table 8.3. Compared to Tetris, Abacus reduces the average movement of the cells during legalization by about 30%, demonstrating that the global placement is better preserved in Abacus. Consequently, the routed wirelength, and the number of vias are decreased by about 1% if Abacus is used. Using Abacus, the runtime of the complete placement process is increased on average by about 6.6%; with Tetris, the runtime is increased by 0.5%. In summary, Abacus gives better results than Tetris and increases the runtime not significantly.

	Kraftwerk				**ROOSTER**				**mPL**			**APlace2**		
Circuit	CPU Place	CPU Rout	rWL	# Vias	CPU Place	CPU Rout	rWL	# Vias	CPU Rout	rWL	# Vias	CPU Rout	rWL	# Vias
ibm01e	16	297	0.678	118482	246	382	0.718	122873	600	0.718	123064	7207	0.790^*	158646
ibm01h	15	354	0.673	119710	242	546	0.725	124063	600	0.691	213162	6606	0.732^*	161717
ibm02e	39	364	1.840	253027	672	546	2.000	256155	600	1.821	250527	491	1.846	254713
ibm02h	32	387	1.977	265587	660	600	1.978	262022	710	1.897	260455	764	1.973	268259
ibm07e	105	551	3.559	469384	1347	710	3.953	470104	1147	4.129	492947	928	3.975	500574
ibm07h	102	591	3.601	483191	1314	1037	4.091	489067	1420	4.240	516929	1255	4.141	518089
ibm08e	132	844	3.993	559984	2096	873	4.231	559010	1256	4.372	579926	983	3.960	595528
ibm08h	141	715	3.926	567249	2063	1037	4.240	577879	1420	4.280	599467	983	3.960	595528
ibm09e	140	582	2.877	484327	1455	600	3.200	473605	938	3.319	488697	600	3.095	502455
ibm09h	127	493	2.890	487189	1424	600	3.205	480961	1037	3.454	502742	655	3.102	512764
ibm10e	175	871	5.660	759409	2312	1146	6.420	755673	1638	6.553	777389	1256	6.178	782942
ibm10h	169	890	5.692	761935	2292	1419	6.544	781897	1801	6.474	799544	1529	6.169	801605
ibm11e	172	670	4.319	629705	1920	819	4.746	613437	1201	4.917	633640	983	4.755	648044
ibm11h	177	650	4.281	629790	1878	873	4.716	625654	1365	4.912	660985	1310	4.818	677455
ibm12e	189	1371	8.344	923900	2745	1638	9.333	930397	3112	10.185	995921	1747	8.599	921454
ibm12h	191	1516	8.351	941797	2691	2129	9.282	942551	2730	9.724	976993	2730	8.814	961296
Average	**1.00**	**1.00**	**1.000**	**1.000**	**14.04**	**1.36**	**1.097**	**1.004**	**1.92**	**1.117**	**1.080**	**4.04**	**1.072**	**1.078**

Table 8.3: Results in the IBM-PLACE 2.0 benchmark suite. *means there are some routing violations. "rWL" is the routed wirelength. "# Vias" means the number of vias.

	Abacus					**Tetris**				
Circuit	CPU Leg	Move	CPU Route	rWL	# Vias	CPU Leg	Move	CPU Route	rWL	# Vias
ibm01e	0.79	0.913	297	0.678	118482	0.12	1.073	292	0.680	120198
ibm01h	0.88	1.175	354	0.673	119710	0.21	1.707	361	0.679	121424
ibm02e	2.43	0.721	364	1.840	253027	0.17	0.888	311	1.859	253170
ibm02h	2.06	0.854	387	1.977	265587	0.27	1.296	472	2.056	271696
ibm07e	6.44	0.542	551	3.559	469384	0.46	0.753	584	3.595	473695
ibm07h	8.66	1.000	591	3.601	483191	1.08	1.488	681	3.705	491398
ibm08e	8.75	0.569	844	3.993	559984	0.45	0.752	640	4.038	568458
ibm08h	9.57	0.579	715	3.926	567249	0.52	0.927	732	3.989	573271
ibm09e	9.80	0.618	582	2.877	484327	0.47	0.956	488	2.901	488415
ibm09h	8.73	0.620	493	2.890	487189	0.65	1.009	510	2.932	490594
ibm10e	11.23	0.543	871	5.660	759409	0.65	0.808	898	5.715	764847
ibm10h	11.35	0.542	890	5.692	761935	0.67	0.823	919	5.738	768437
ibm11e	12.52	0.536	670	4.319	629705	0.58	0.794	680	4.348	633766
ibm11h	13.07	0.554	650	4.281	629790	0.73	0.879	723	4.323	633421
ibm12e	11.16	0.535	1371	8.344	923900	0.64	0.748	1211	8.409	930654
ibm12h	11.36	0.541	1516	8.351	941797	0.66	0.794	1371	8.384	941651
Average	**6.6%**$^+$	**1.000**	**1.00**	**1.000**	**1.000**	**0.5%**$^+$	**1.456**	**0.995**	**1.012**	**1.010**

Table 8.4: Results in the IBM-PLACE 2.0 benchmark suite. Comparison between Abacus and Tetris for legalization. “CPU Leg” is the runtime of legalization. $^+$means the ratio between the runtime of legalization and the runtime of the complete placement process. “Move” is the average cell movement during legalization, normalized to the average cell dimension of each circuit. “rWL” is the routed wirelength. “# Vias” means the number of vias.

8.4 ISPD 2006 Contest Benchmark Suite

The ISPD 2006 contest benchmark suite was introduced in an international placement contest [ISP06] and consists of eight circuits with up to 2.5 million movable modules. The quality of a placer is measured based on three parameters: the netlength in HPWL, the CPU factor and an overflow factor. The overflow factor is zero if the given upper limit d_{up} for the module density is respected everywhere on the chip. Thus, the overflow factor, in combination with a low d_{up}, should assure routability. The CPU factor is derived from the logarithmic ratio between the placer’s CPU time and the median over the CPU times of all placers, which completed this benchmark suite. For example, a CPU factor of –4% (+4%) represents that the placer’s CPU time is two times smaller (greater) than the median CPU time. The three parameters are combined in three quality metrics: HPWL, HPWL+Overflow, and HPWL+Overflow+CPU. The last quality metric considers routability and runtime and was deciding in the placement contest. In the following, all three quality metrics are normalized to the best values published in [ISP06].

Table 8.5 shows the detailed results of Kraftwerk. The low overflow factor of 1.87% demonstrates that Kraftwerk respects the upper limit d_{up} of the module density very well. Therefore, the control of the module density (with $td = d_{up}$), described in Section 5.11, is very effective. The very low CPU factor of –9.35% reveals that runtimes of Kraftwerk are more than four times smaller than the median runtime. To obtain the CPU factor, the runtimes of Kraftwerk are scaled in Table 8.6 (a) with 0.86, since the results of [DES] (which are used for normalization) are based on a different machine.

Table 8.6 summarizes the results of Kraftwerk and of other state-of-the-art placers. The results of NTUPlace3 are taken from [CJH+06], using a CPU scaling of 1.1. The results of FastPlace3 are taken from [VPC07], and the CPU scaling is 1.2. The results of RQL are taken from [VNA+07] with a CPU scaling of 1.2. For other placers, the original results [ISP06] of the placement contest are used. Unfortunately, there are no runtimes available of RQL. Based on the CPU factor, Kraftwerk is the fastest placer. According to the main quality metric HPWL+Overflow+CPU, Kraftwerk is the best placer. NTUPlace3 is the second best and has a 3.9% higher value in this quality metric. Ignoring the CPU factor and using the quality metric HPWL+Overflow, Kraftwerk is the fourth best. NTUPlace3, RQL, and mPL6 are 4.1%, 3.0%, and 2.9% better, respectively. Unfortunately, there are no recent results of FastPlace3 in HPWL and HPWL+Overflow available. The same holds true for recent results of RQL in HPWL+Overflow+CPU.

In summary, Table 8.6 reveals that Kraftwerk offers excellent results in extreme low runtime. The same holds true for the original results of Kraftwerk in the placement contest. The presented results demonstrate the efficiency of various features of Kraftwerk. For example, using the Bound2Bound net model to express the HPWL netlength accurately in the cost function, or using the advanced methods for the module demand and module supply to prevent halos around large modules and to control the module density.

					Score		
Circuit	HPWL	Overflow factor	CPU	CPU factor	HPWL	HPWL+ Overflow	HPWL+ Overflow+ CPU
adaptec5	433.84	3.606%	1618	– 9.35%	1.071	1.032	0.939
newblue1	65.92	0.415%	603	– 8.38%	1.057	1.043	0.956
newblue2	203.91	1.286%	508	– 10.00%*	1.033	1.082	0.975
newblue3	278.51	0.382%	526	– 10.00%*	1.018	1.067	0.961
newblue4	304.24	1.709%	1553	– 8.63%	1.068	1.033	0.945
newblue5	548.38	2.694%	2622	– 9.50%	1.109	1.054	0.957
newblue6	528.59	1.702%	2579	– 9.89%	1.048	1.036	0.936
newblue7	1126.58	3.155%	4828	– 9.06%	1.053	1.051	0.958
Average		**1.869%**		**– 9.35%**	**1.057**	**1.050**	**0.953**

Table 8.5: Results of Kraftwerk in the ISPD 2006 contest benchmark suite. *As required in this benchmark suite, the CPU factor is limited to ±10%. The “raw” CPU-factors are –13.50% and –10.98%, respectively.

			Score		
Placer	Overflow factor	CPU factor	HPWL	HPWL+ Overflow	HPWL+ Overflow+ CPU
Kraftwerk	1.87 %	– 9.35 %	1.057	1.050	0.953
NTUPlace3	6.26 %	– 2.61 %	0.976	1.007	0.990
RQL	6.80 %	n.a. %	0.981	1.018	n.a.
Fastplace3	n.a.	– 8.17 %	n.a.	n.a.	1.040
mPL6	1.36 %	1.58 %	1.035	1.020	1.040
mFAR	2.71 %	– 0.12 %	1.108	1.107	1.108
APlace3	3.83 %	5.31 %	1.097	1.107	1.165
Dragon	0.12 %	– 5.90 %	1.331	1.300	1.232
DPlace	9.32 %	– 4.54 %	1.343	1.414	1.364
Capo	0.32 %	2.69 %	1.375	1.344	1.385

Table 8.6: Results of various placers in the ISPD 2006 contest benchmark suite.

8.5 ISPD 2005 Contest Benchmark Suite

Similar to the previous presented benchmark suite, the ISPD 2005 contest benchmark suite [ISP05, NAV$^+$05] was also introduced in an international placement contest. The suite consists of eight circuits with up to 2.2 million movable modules. The quality of placement is measured by the HPWL netlength. Routability is ignored completely in this benchmark suite. Table 8.7 depicts the results of Kraftwerk and other state-of-the-art placers. The results of NTUPlace3 are taken from [CJH$^+$06], using a CPU scaling of 1.1. The results of FastPlace3 are taken from [VPC07], and the CPU scaling is 1.2. The results of RQL are taken from [VNA$^+$07] with a CPU scaling of 1.2. The results of other placers are taken from [KRW05]. Unfortunately, in [KRW05], there are no detailed runtimes published, and no results for the circuits adaptec1 and adaptec3 are published. On average, Kraftwerk is as good as FastPlace3 in netlength, but two times faster. Compared with RQL, Kraftwerk has a 5.38% higher netlength, but is more than three times faster. Compared with NTUPlace3, Kraftwerk has a 2.2% higher netlength, but is more than three times faster. Relative to APlace2, Kraftwerk has a 3.5% higher netlength, but is almost fourty times faster. According to the netlength of the remaining other placers, Kraftwerk is between 2.7% and 30% better. Hence, the results in the ISPD 2005 contest benchmark suite benchmark show that Kraftwerk is a fast placer, which offers comparable results in the HPWL netlength. The open question here is how relevant a low HPWL netlength is, if routability is not considered. In the ISPD 2006 contest benchmark suite, which is successor of the ISPD 2005 contest benchmark suite, routability is considered by setting an upper limit for the module density. Results of the ISPD 2006 contest benchmark suite are presented in the previous section.

	Kraftwerk		**FastPlace3**		**RQL**		**NTUPlace3**		**APlace2**	**mFAR**	**Dragon**	**mPL5**	**Capo**
Circuit	HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	HPWL	HPWL	HPWL	HPWL
adaptec1	82.43	262	79.38	353	77.82	751	80.93	883	n.a.	n.a.	n.a.	n.a.	n.a.
adaptec2	92.85	349	93.08	559	88.51	1247	89.85	906	87.31	91.53	94.72	97.11	99.71
adaptec3	227.22	713	217.80	2275	210.96	2405	214.20	1944	n.a.	n.a.	n.a.	n.a.	n.a.
adaptec4	199.43	709	201.36	1411	188.86	2096	193.74	2325	187.65	190.84	200.88	200.94	211.25
bigblue1	97.67	407	95.68	604	94.98	1160	97.28	1675	94.64	97.70	102.39	98.31	108.21
bigblue2	154.74	559	155.10	1380	150.03	2261	152.20	3352	143.82	168.70	159.71	173.22	172.30
bigblue3	343.32	2070	379.88	4642	323.09	4864	348.48	6256	357.89	379.95	380.45	369.66	382.63
bigblue4	852.40	4147	832.88	6862	797.66	12410	829.16	11308	833.21	876.28	903.96	904.19	1098.76
Average	**1.000**	**1.00**	**1.000**	**2.00**	**0.959**	**3.12**	**0.979**	**3.48**	**0.967**	**1.028**	**1.046**	**1.053**	**1.126**

Table 8.7: Results in the ISPD 2005 contest benchmark suite.

8.6 ICCAD 2004 Mixed-Size Benchmark Suite

The ICCAD 2004 mixed size benchmark suite [ACaR$^+$04] consists of eighteen circuits with up to 200k movable modules. The number of macros is about 400 per circuit. Table 8.8 summarizes the results of Kraftwerk and of other placers in this benchmark suite. Results of FDP are taken from [VK05b] with a CPU scaling of 1.1. Results of APlace2 and mPL5 are taken from [CJH$^+$06] with a CPU scaling of 1.1. Results of NTUPlace3 are taken from [CJH$^+$06], using a CPU scaling of 1.1. Kraftwerk is the fastest placer, ranging from 3.52 faster than NTUPlace3 up to 24 times faster than APlace2. In the HPWL netlength, Kraftwerk is 1.0%, and 5.3% better than mPL5, and FDP, respectively. Compared to APlace2, and NTUPlace3, Kraftwerk has a 0.5%, and 1.8% higher netlength, respectively. The results in the ICCAD 2004 mixed size benchmark suite demonstrates that Kraftwerk is a fast placer, which offers good results. With these results, also the efficiency of different features of Kraftwerk are shown. Amongst others, using a move force proportional to the module area, the macros are moved away from the standard cells, and standard cells are moved a small distance during global placement, which improves the netlength. Using Puzzle with Tabu Search, big macros are legalized with minimal total movement.

8.7 IBM-HB$^+$ Floorplacement Benchmark Suite

The IBM-HB$^+$ floorplacement benchmark suite [NARM06] consists of seventeen circuits, and is derived from the same benchmark suite (IBM/ISPD'98) as the ICCAD 2004 mixed size benchmark suite. However, the IBM-HB$^+$ circuits do not have standard cells, but consist of about 1000 macros with various dimensions. The dimensions of the macros are fixed, and the placement area is given. Therefore, this benchmark suite is called "floorplacement" in [NARM06, RAPM06]. Since a big part of the placement area is occupied by just a few macros, and there is little free space in the placement area, the circuits are considered as hard instances in [NARM06]. In addition, only results of SCAMPI are available in [NARM06]. Other placers produce invalid placements, in which some macros overlap, or not all macros

Circuit	**Kraftwerk**		**FDP**		**NTUPlace3**		**APlace2**		**mPL5**	
	HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	CPU	HPWL	CPU
ibm01	2.24	11	2.42	145	2.17	33	2.14	381	2.22	91
ibm02	4.90	27	5.11	284	4.63	63	4.65	872	4.68	264
ibm03	6.61	24	7.08	337	6.65	72	6.71	1015	6.86	300
ibm04	7.63	29	7.69	317	7.21	89	7.57	977	7.69	261
ibm05	9.79	33	n.a.	n.a.	9.66	160	9.69	766	10.09	130
ibm06	6.11	40	6.20	389	5.94	95	6.02	967	6.16	520
ibm07	10.42	52	10.57	607	9.90	219	10.00	1296	9.96	692
ibm08	12.97	85	13.30	719	12.29	235	12.50	1484	11.92	1133
ibm09	11.98	71	13.30	713	12.00	213	12.13	1837	13.15	1363
ibm10	30.15	232	30.70	924	28.49	351	28.83	2649	29.36	1654
ibm11	17.59	107	18.41	950	17.54	336	18.67	3814	17.87	1071
ibm12	31.42	124	36.46	1472	32.07	332	33.42	3663	33.43	1419
ibm13	22.48	147	23.60	1175	22.16	536	22.80	3845	22.52	1079
ibm14	35.13	308	37.84	2185	35.36	1274	35.92	4723	34.99	1588
ibm15	47.58	468	47.69	2468	45.38	1251	46.81	5419	50.88	4989
ibm16	54.17	527	61.27	2792	57.59	1595	54.53	6109	55.21	6200
ibm17	66.63	474	69.45	3577	66.73	2123	65.67	6635	66.96	2131
ibm18	42.36	609	44.88	4369	41.58	2874	41.99	10925	43.99	2477
Average	**1.000**	**1.00**	**1.056**	**9.02**	**0.982**	**3.25**	**0.995**	**23.93**	**1.010**	**9.67**

Table 8.8: Results in ICCAD 2004 mixed size benchmark suite.

are within the placement region. In contrast to this, all placements of Kraftwerk (and of SCAMPI) are valid. Compared to SCAMPI, Kraftwerk has a 14% better HPWL netlength, and is about eight times faster. In Kraftwerk, the legalization is done with Puzzle (without using Tabu Search). Hence, the excellent results of Kraftwerk in this benchmark suite reveals, amongst others, the efficiency of Puzzle. In addition, the results demonstrate that Kraftwerk is a robust placer, which can even place such hard instances.

8.8 Average-Case Computational Complexity

Figure 8.1 displays the runtimes of Kraftwerk versus the number N of movable modules. The results are obtained by placing the ISPD 2005/2006 contest benchmark suites. The average-case computations complexity is $\Theta(N^{1.18})$, and thus nearly linear. Hence, Kraftwerk can easily cope with future circuits having an increasement in N.

Circuit	**Kraftwerk**		**SCAMPI**	
	HPWL	CPU	HPWL	CPU
ibm-HB$^+$01	2.83	10	3.4	68
ibm-HB$^+$02	5.88	25	8.0	154
ibm-HB$^+$03	9.23	16	9.5	115
ibm-HB$^+$04	10.02	18	12.3	158
ibm-HB$^+$06	10.76	12	11.0	187
ibm-HB$^+$07	14.93	16	15.7	110
ibm-HB$^+$08	21.01	22	20.5	207
ibm-HB$^+$09	17.50	18	22.2	200
ibm-HB$^+$10	45.71	53	55.2	351
ibm-HB$^+$11	25.77	23	27.8	159
ibm-HB$^+$12	51.29	43	67.6	447
ibm-HB$^+$13	34.85	23	42.2	231
ibm-HB$^+$14	63.08	42	66.4	295
ibm-HB$^+$15	92.36	46	88.2	414
ibm-HB$^+$16	95.62	54	106.2	337
ibm-HB$^+$17	148.16	99	152.7	424
ibm-HB$^+$18	74.44	53	77.8	211
Average	**1.000**	**1.00**	**1.140**	**7.99**

Table 8.9: Results in IBM-HB$^+$ floorplacement benchmark suite.

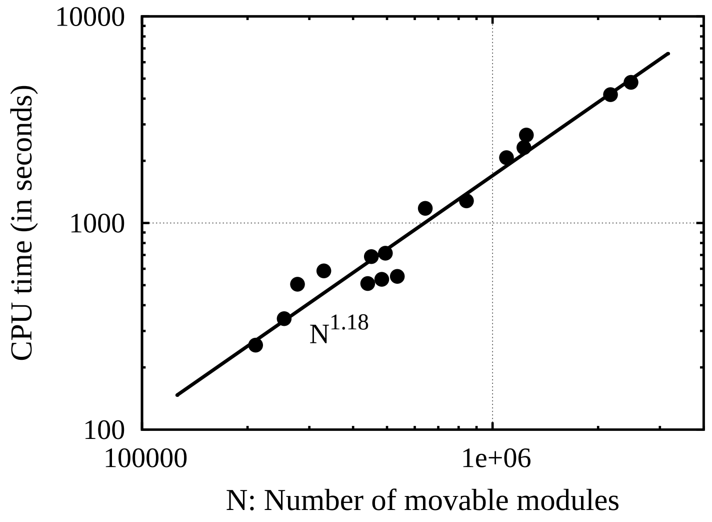


Figure 8.1: Average-case computational complexity of Kraftwerk.

Chapter 9

Conclusion

Integrated circuits play an important role in industry, and in our daily life. To cope with the complexity, and to lower the design time, integrated circuits are designed by computer algorithms today. This design process is called EDA (electronic design automation), and consists of several consecutive steps. One key step is the layout synthesis, as it highly affects the quality of the circuit. Starting from a gate level description, layout synthesis means to place the modules (placement) and to route the nets (routing). After this, the polygon level is reached, and the circuit can be fabricated.

This thesis presents novel approaches for all main steps of placement. Each step is driven by expressing the objective in a quadratic cost function, which can be minimized efficiently. During global placement, netlength and routability are optimized. Legalization then removes the remaining module overlap of global placement and targets the module movement. The key features of the placement approaches presented in this thesis are as follows:

- Kraftwerk is a global placer. It is driven by a generic demand-and-supply system, and utilizes two forces to spread the modules over the placement area. Both forces are determined and modeled in a systematic way. As a consequence, Kraftwerk converges such that the demand is adapted further to the supply in each placement iteration, which in principle means that the module overlap is reduced in each placement iteration.

- Due to the generic demand-and-supply system and the systematic force modeling, Kraftwerk is versatile, robust, stable, and fast. Versatile, because of the demand-and-supply system, different placement types are supported (e.g., standard cell circuits, macro cell circuits, mixed-size circuits, and circuits with fixed modules). Furthermore, various objectives (e.g., routability) can be considered in addition to minimal netlength. Kraftwerk is robust, because it successfully places even hard instances of placement, e.g., placing some big modules in a narrow placement area. Kraftwerk is stable, because for small changes in the circuit, the changes in the placement are also small. Kraftwerk is fast, because the runtime is extremely low.

- The Bound2Bound net model enables the accurate representation of the HPWL netlength in the quadratic cost function. Consequently, the obtained placements are excellent in the HPWL netlength. In addition, experiments on routability-driven placement reveal that the HPWL metric is a sufficient estimation of the routed wirelength.

- RUDY is fast and accurate routing demand estimation approach. It is integrated in the demand-and-supply system of Kraftwerk, in order to optimize routability of a circuit during global placement.
- Puzzle is a legalization approach, suitable for macro cell circuits. For each overlapping macro pair, the overlap is removed either in x or y direction. Initially, the directions are determined based on a given placement. In addition, Tabu Search is used to optimize the directions, and thus to reduce the movement of the macros during legalization.
- Abacus is a fast and greedy legalization approach, applicable to align standard cells to a given row structure. Cells within one row are placed by dynamic programming. Already legalized cells are moved, which reduces the total movement of all cells.

The presented experimental results demonstrate that the described placement approaches give high quality placements in extremely low runtime. With an almost linear average-case computational complexity, the approaches are applicable for future circuits with an increasing complexity.

Bibliography

[AA88] A. Alon, U. Ascher: *Model and solution strategy for placement of rectangular blocks in the euclidean plane*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-7(3), pages 378–386, March 1988.

[ABD$^+$07] P. Azzoni, M. Bertoletti, N. Dragone, F. Fummi, C. Guardiani, W. Vendraminetto: *Yield-aware placement optimization*, in: *Design, Automation and Test in Europe (DATE)*, pages 1232 – 1237, 2007.

[ACaR$^+$04] S. N. Adya, S. Chaturvedi, J. a Roy, D. A. Papa, I. L. Markov: *Unification of partitioning, placement and floorplanning*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 550–557, 2004.

[ACH$^+$97] C. Alpert, T. Chan, D. J.-H. Huang, I. Markov, K. Yan: *Quadratic placement revisited*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 752–757, 1997.

[AHK97] C. Alpert, J.-H. Huang, A. B. Kahng: *Multilevel circuit partitioning*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 530–533, 1997.

[AJK82] K. J. Antreich, F. M. Johannes, F. H. Kirsch: *A new approach for solving the placement problem using force models*, in: *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 481–486, 1982.

[AK89] E. Aarts, J. Korst: *Simulated annealing and boltzmann machines - a stochastic approach to combinatorial optimization and neural computing*, 1989.

[AM07] A. R. Agnihotri, P. H. Madden: *Fast analytic placement using minimum cost flow*, in: *Asia and South Pacific Design Automation Conference*, pages 128–134, 2007.

[And74] T. Anderson: *An Introduction To Multivariate Statistical Analysis*, John Wiley & Sons, Inc., 1974.

[ANVY05] C. J. Alpert, G.-J. Nam, P. Villarribua, M. C. Yildiz: *Placement stability metrics*, in: *Asia and South Pacific Design Automation Conference*, pages 1144 – 1147, 2005.

[AOL$^+$05] A. R. Agnihorti, S. Ono, C. Li, M. C. Yildiz, A. Khathate, C.-K. Koh, P. H. Madden: *Mixed block placement via fractional cut recursive bisection*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 24(5), pages 748–761, May 2005.

[Apt90] J. Apte: *A Layout Automation Problem Combining Standard Cells and Macro Blocks*, Ph.D. thesis, Department of Computer Science, Duke University, Durham, USA, 1990.

[BH83] M. Burstein, S. J. Hong: *Hierarchical VLSI layout: Simultaneous placement and wiring of gate-arrays*, in: *VLSI '83, Proc. of the IFIP TC 10 / WG 10.5 Int. Conf. on Very Large Scale Integration*, pages 45–60, North-Holland, Amsterdam, New York, Oxford, August 1983.

[BJ90] M. R. Berkelaar, J. A. Jess: *Gate sizing in mos digital circuits with linear programming*, in: *Design, Automation and Test in Europe (DATE)*, pages 217 – 221, 1990.

[BKT93] K. D. Boese, A. B. Kahng, C. A. Tsao: *Best-so-far vs. where-you-are: New perspectives on simulated annealing for CAD*, in: *European Design Automation Conference with EURO-VHDL (EURO-DAC)*, pages 78–83, 1993.

[Bla85a] J. Blanks: *Near-optimal placement using a quadratic objective function*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 22, pages 609–615, 1985.

[Bla85b] J. Blanks: *Near-optimal quadratic-based placement for a class of IC layout problems*, ieeecircuitsdevices, pages 31–37, September 1985.

[BPV04] U. Brenner, A. Pauli, J. Vygen: *Almost optimum placement legalization by minimum cost flow and dynamic programming*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 2–9, 2004.

[BR02] U. Brenner, A. Rohe: *An effective congestion driven placement framework*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 6–11, 2002.

[Bre77a] M. Breuer: *A class of min-cut placement algorithms*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 14, pages 284–290, 1977.

[Bre77b] M. Breuer: *Min-cut placement*, Journal of Design Automation and Fault Tolerant Computing, volume 1, pages 343–362, 1977.

[BS05] U. Brenner, M. Struzyna: *Faster and better global placement by a new transportation algorithm*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 591–596, 2005.

[BV00] U. Brenner, J. Vygen: *Faster optimal single-row placement with fixed ordering*, in: *Design, Automation and Test in Europe (DATE)*, pages 117–121, 2000.

[BV04] U. Brenner, J. Vygen: *Legalizing a placement with minimum total movement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 23(12), pages 1597–1613, December 2004.

[CCPY02] C.-C. Chang, J. Cong, Z. D. Pan, X. Yuan: *Physical hierarchy generation with routing congestion control*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 36–41, 2002.

[CCS05] T. Chan, J. Cong, K. Sze: *Multilevel generalized force-directed method for circuit placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 185–192, 2005.

[CCWW00] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, S.-W. Wu: *B*-trees: A new representation for non-slicing floorplans*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 37, pages 458–463, 2000.

[CCY03] C.-C. Chang, J. Cong, X. Yuan: *Multi-level placement for large-scale mixed-size ic designs*, in: *Asia and South Pacific Design Automation Conference*, 325-330, 2003.

[cCYc$^+$07] T. chieh Chen, P.-H. Yuh, Y.-W. chang, F.-J. Huang, D. Liu: *MP-trees: A packing-based macro placement algorithm for mixed-size designs*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 447–452, 2007.

[Chu04] C. Chu: *FLUTE: Fast lookup table based wirelength estimation technique*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 696–701, 2004.

[CJH$^+$06] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, Y.-W. Chang: *A high-quality mixed-size analytical placer considering preplaced blocks and density constraints*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 187–192, 2006.

[CK83] C. Cheng, E. Kuh: *Partitioning and placement based on network optimization*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 86–87, 1983.

[CK84] C.-K. Cheng, E. S. Kuh: *Module placement based on resistive network optimization*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-3(3), pages 218–225, July 1984.

[CKM00] A. E. Caldwell, A. B. Kahng, I. L. Markov: *Optimal partitioners and end-case placers for standard-cell layout*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 19(11), pages 1304–1313, November 2000.

[CLL$^+$97] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, D. Xu: *Large scale circuit partitioning with loose/stable net removal and signal flow based clustering*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 441–446, 1997.

[Cor] S. P. E. Corporation: *SPEC CPU 2000*, `http://www.spec.org/cpu2000`.

[Cor79] L. I. Corrigan: *A placement capability based on partitioning*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 16, pages 406–413, 1979.

[CP80] L. Coté, A. Patel: *The interchange algorithms for circuit placement problems*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 17, pages 528–534, 1980.

[CP86] J. Cohoon, W. Paris: *Genetic placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 374–377, November 1986.

[CP87] J. P. Cohoon, W. P. Paris: *Genetic placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-6(6), pages 956–964, November 1987.

[CRS88] J. P. Cohoon, D. S. Richards, J. S. Salowe: *A linear-time steiner tree routing algorithm for terminals on the boundary of a rectangle*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 402–405, 1988.

[CS07] P. Chong, C. Szegedy: *A morphing approach to address placement stability*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 95–102, 2007.

[CX06] J. Cong, M. Xie: *A robust detailed placement for mixed-size ic design*, in: *Asia and South Pacific Design Automation Conference*, pages 188–194, 2006.

[DD96a] S. Dutt, W. Deng: *A probability-based approach to vlsi circuit partitioning*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 100–105, 1996.

[DD96b] S. Dutt, W. Deng: *VLSI circuit partitioning by cluster-removal using iterative improvement techniques*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1996.

[DES] *International symposium on physical design*, `http://www.ispd.cc`.

[DJA94] K. Doll, F. M. Johannes, K. J. Antreich: *Iterative placement improvement by network flow methods*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 13(10), pages 1189–1200, October 1994.

[DK83] A. E. Dunlop, B. W. Kernighan: *A placement procedure for polycell VLSI circuits*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 51–52, 1983.

[DK85] A. Dunlop, B. Kernighan: *A procedure for placement of standard-cell VLSI circuits*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-4(1), pages 92–98, January 1985.

[DK87] W.-M. Dai, E. S. Kuh: *Global spacing of building-block layout*, vlsi, pages 193–205, 1987.

[Don80] W. Donath: *Complexity theory and design automation*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 19, pages 412–419, 1980.

[Eis97] J. Eisner: *State-of-the-art algorithms for minimum spanning trees: A tutorial discussion*, Technical report, University of Pennsylvania, April 1997.

[Eis99] H. Eisenmann: *Ein universelles Plazierverfahren für integrierte Schaltungen*, Ph.D. thesis, June 1999.

[EJ98] H. Eisenmann, F. M. Johannes: *Generic global placement and floorplanning*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 269–274, June 1998.

[EK97] H. Esbensen, E. S. Kuh: *A performance-driven IC/MCM placement algorithm featuring explicit design space exploration*, ACM Transactions on Design Automation of Electronic Systems, volume 2(1), pages 62–80, 1997.

[FCW67] C. J. Fisk, D. L. Caskey, L. E. West: *Accel: Automated circuit card etching layout*, Proceedings of the IEEE, volume 55(11), pages 1971–1982, November 1967.

[FK86] J. Frankle, R. Karp: *Circuit placements and cost bounds by eigenvector decomposition*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 414–417, November 1986.

[FM82] C. Fiduccia, R. Mattheyses: *A linear-time heuristic for improving network partitions*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 19, pages 175–181, 1982.

[For87] R. Forbes: *Heuristic acceleration of force-directed placement*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 24, pages 735–740, 1987.

[FYSK83] K. Fukunaga, S. Yamada, H. S. Stone, T. Kasai: *Placement of circuit modules using a graph space approach*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 20, pages 465–471, 1983.

[GB83] M. Goldberg, M. Burstein: *Heuristic improvement technique for bisection of VLSI networks*, in: *IEEE International Conference on Computer Design (ICCD)*, pages 122–125, 1983.

[GGL] *GLPK: GNU linear programming kit*, http://www.gnu.org/software/glpk/.

[GJ77] M. R. Garey, D. S. Johnson: *The rectilinear steiner tree is NP-complete*, SIAM Journal of Applied Mathematics, volume 32(4), pages 826–834, April 1977.

[GJ79] M. R. Garey, D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, New Jersey, 1979.

[GKP05] P. Gupta, A. B. Kahng, C.-H. Park: *Detailed placement for improved depth of focus and cd control*, in: *Asia and South Pacific Design Automation Conference*, pages 343 – 348, 2005.

[GL97] F. Glover, M. Laguna: *Tabu Search*, Springer, 1997.

[Got79] S. Goto: *A two-dimensional placement algorithm for the master slice LSI layout problem*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 16, pages 11–17, 1979.

[Got81] S. Goto: *An efficient algorithm for the two-dimensional placement problem in electrical circuit layout*, IEEE Transactions on Circuits and Systems CAS, volume CAS-28(1), pages 12–18, January 1981.

[GRSZ94] J. Griffith, G. Robins, J. S. Salowe, T. Zhang: *Closing the gap: Near-optimal steiner trees in polynomial time*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 13(11), pages 1351, 11 1994.

[GW03] M. Gertz, S. Wright: *Object-oriented software for quadratic programming*, ACM Transactions on Mathematical Software, volume 29(1), pages 58–81, March 2003.

[Hal70] K. M. Hall: *An r-dimensional quadratic placement algorithm*, Management Science, volume 17(3), pages 219–229, November 1970.

[Han66] M. Hanan: *On Steiner's Problem with Rectiliner Distance*, SIAM Journal of Applied Mathemetics, volume 14(2), pages 255–265, 1966.

[HCC92] T. Hamada, C.-K. Cheng, P. M. Chau: *An efficient multilevel placement technique using hierarchical partitioning*, IEEE Transactions on Circuits and Systems CAS, volume CAS-39(6), pages 432–439, July 1992.

[Hil02] D. Hill: *Method and system for high speed detailed placement of cells within integrated circuit designs*, U.S. Patent 6370673, April 2002.

[HK72] M. Hanan, J. Kutzberg: *Placement techniques*, in: M. Breuer (ed.), *Design Automation of Digital Systems, Volume 1: Theory and Techniques*, pages 213–282, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

[HK97] D. J.-H. Huang, A. B. Kahng: *Partitioning-based standard-cell global placement with an exact objective*, in: *International Symposium on Physical Design (ISPD)*, pages 18–25, 1997.

[HL00] S.-W. Hur, J. Lillis: *Mongrel: Hybrid techniques for standard cell placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 165–170, 2000.

[HMS02a] B. Hu, M. Marek-Sadowska: *Congestion minimization during placement without estimation*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 739–745, 2002.

[HMS02b] B. Hu, M. Marek-Sadowska: *FAR: Fixed-points addition & relaxation based placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 161–166, 2002.

[HMS05] B. Hu, M. Marek-Sadowska: *Multilevel fixed-point-addition-based vlsi placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 24(8), pages 1188–1203, August 2005.

[HRSV86] M. Huang, F. Romeo, A. Sangiovanni-Vincentelli: *An efficient general cooling schedule for simulated annealing*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 381–384, 1986.

[HVW90] J. Ho, G. Vijayan, C. Wong: *New algorithms for the rectilinear steiner tree problem*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-9(2), pages 185–193, February 1990.

[HW76] G. Hall, J. M. Watt: *Modern numerical methods for ordinary differential equations*, Claredon Press, Oxford, 1976.

[HWA76] M. Hanan, P. K. Wolff, Sr., B. J. Agule: *Some experimental results on placement techniques*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 13, pages 214–224, 1976.

[Hwa79] F. K. Hwang: *An O(nlogn) algorithm for suboptimal rectilinear steiner trees*, IEEE Transactions on Circuits and Systems CAS, volume CAS-26(1), pages 75–77, January 1979.

[HWM86] H. Hillner, B. X. Weis, D. A. Mlynski: *The discrete placement problem: A dynamic programming approach*, in: *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 315–318, 1986.

[HYH$^+$01] W. Hou, H. Yu, X. Hong, Y. Cai, W. Wu, J. Gu, W. H. Kao: *A new congestion-driven placement algorithm based on cell inflation*, in: *Asia and South Pacific Design Automation Conference*, pages 605–608, 2001.

[IKB83] A. Iosupovici, C. King, M. Breuer: *A module interchange placement machine*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 20, pages 171–174, 1983.

[ISP05] *ISPD 2005 placement contest*, `http://www.sigda.org/ispd2005/contest.htm`, March 2005.

[ISP06] *ISPD 2006 placement contest*, `http://www.sigda.org/ispd2006/contest.html`, March 2006.

[JJA83] F. M. Johannes, K. M. Just, K. J. Antreich: *On the force placement of logic arrays*, in: *Proceedings European Conference on Circuit Theory and Design (ECCTD)*, pages 203–206, 1983.

[JK89] M. A. B. Jackson, E. S. Kuh: *Performance-driven placement of cell based IC's*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 26, pages 370–375, 1989.

[Joh87] F. Johannes: *Use of triangulation for global placement*, vlsi, pages 183–191, 1987.

[Jus87] K. M. Just: *Zur automatischen Plazierung der Moduln bei der Layout-Synthese*, Ph.D. thesis, 1987.

[KAKS97] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar: *Multilevel hypergraph partitioning: Application in VLSI domain*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 526–529, 1997.

[KB89] R.-M. Kling, P. Banerjee: *ESP: Placement by simulated evolution*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-8(3), pages 245–256, March 1989.

[KB91] R. M. Kling, P. Banerjee: *Empirical and theoretical studies of the simulated evolution method applied to standard cell placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-10(10), pages 1303–1315, October 1991.

[KGV83] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi: *Optimization by simulated annealing*, Science, volume 220, pages 671–680, 1983.

[Kie53] J. Kiefer: *Sequential minimax search for a maximum*, in: *Proceedings of the American Mathematical Society*, volume 4 of *3*, pages 502–506, June 1953.

[Kir82] F. H. Kirsch: *Ein Lösungsverfahren zur Plazierung von Bauelementen mittels eines Kräftemodells*, Archiv für Elektronik und Übertragungstechnik (AEÜ), volume 36(10), pages 393–401, 1982.

[Kir84] F. Kirsch: *Rechnergestützte Lösungsverfahren zur Relativplazierung bei der Layoutsynthese*, Ph.D. thesis, 1984.

[KK92] S.-S. Kim, C.-M. Kyung: *Circuit placement on arbitrarily shaped regions using the self-organization principle*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-11(7), pages 844–853, July 1992.

[KKM91] C.-M. Kyung, P. Kraus, D. Mlynski: *An analytic algorithm for global circuit placement*, integration, volume 11(2), pages 191–204, April 1991.

[KL70] B. Kernighan, S. Lin: *An efficient heuristic procedure for partitioning graphs*, The Bell Systems Technical Journal, volume 49, pages 291–307, February 1970.

[KLA$^+$04] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, P. H. Madden: *Recursive bisection based mixed block placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 84–89, 2004.

[Kle77] V. Klee: *Can the measure of $\cup[a_i, b_i]$ be computed in less than o(nlogn) steps?*, American Mathematical Monthly, volume 84, pages 284–285, 1977.

[Kle89] J. M. Kleinhans: *Ein Plazierungsverfahren für den zellenbasierten Layoutentwurf hochintegrierter Schaltungen*, Ph.D. thesis, 1989.

[KMR04] A. B. Kahng, I. L. Markov, S. Reda: *On legalization of row-based placements*, in: *Great Lakes Symposium on VLSI (GLS-VLSI)*, pages 214–219, 2004.

[KP77] K. H. Khokhani, A. M. Patel: *The chip layout problem: A placement procedure for LSI*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 14, pages 291–297, 1977.

[Kri84] B. Krishnamurthy: *An improved min-cut algorithm for partitioning VLSI networks*, IEEE Transactions on Computers, volume C-33(5), pages 438–446, May 1984.

[KRW05] A. B. Kahng, S. Reda, Q. Wang: *Architecture and details of a high quality, large-scale analytical placer*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 890–897, 2005.

[KSJ88] J. M. Kleinhans, G. Sigl, F. M. Johannes: *GORDIAN: A new global optimization / rectangle dissection method for cell placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 506–509, November 1988.

[KSJ89] J. M. Kleinhans, G. Sigl, F. M. Johannes: *Sea-of-gates placement by simultaneous quadratic programming combined with improved partitioning*, in: *IFIP International Conference on Very Large Scale Integration (VLSI)*, pages 445–454, München, 1989.

[KSJA91] J. M. Kleinhans, G. Sigl, F. M. Johannes, K. J. Antreich: *GORDIAN: VLSI placement by quadratic programming and slicing optimization*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 10(3), pages 356–365, March 1991.

[KTZ99] A. B. Kahng, P. Tucker, A. Zelikovsky: *Optimization of linear placements for wirelength minimization with free sites*, in: *Asia and South Pacific Design Automation Conference*, pages 241–244, 1999.

[KV06] A. Kennings, K. P. Vorwerk: *Force-directed methods for generic placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 25(10), pages 2076–2087, October 2006.

[KW01] M. Kowarschik, C. Weiß: *DiMEPACK — a cache-optimized multigrid library*, in: H. Arabnia (ed.), *International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 425–430, CSREA Press, June 2001.

[KW05a] A. B. Kahng, Q. Wang: *Implementation and extensibility of an analytic placer*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 24(05), pages 734–747, May 2005.

[KW05b] A. B. Kahng, Q. Wang: *Implementation and extensibility of an analytic placer*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 24(05), pages 734–747, May 2005.

[KX03] A. B. Kahng, X. Xu: *Accurate pseudo-constructive wirelength and congestion estimation*, in: *International Workshop on System Level Interconnect Prediction*, pages 61–68, 2003.

[Lau79] U. Lauther: *A min-cut placement algorithm for general cell assemblies based on a graph representation*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 16, pages 1–10, 1979.

[LD86] D. LaPotin, S. Director: *Mason: A global floorplanning approach for VLSI design*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-5(4), pages 477–489, October 1986.

[LD88] J. Lam, J. Delosme: *Performance of a new annealing schedule*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 25, pages 306–311, 1988.

[lEC94] C. liang Eric Cheng: *RISA: Accurate and efficient placement routability modeling*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 690–695, 1994.

[Lee93] T.-C. Lee: *A bounded 2d contour searching algorithm for floorplan design with arbitrarily shaped rectilinear and soft modules*, in: *ACM/IEEE Design Automation Conference (DAC)*, page 525, 1993.

[Len88] T. Lengauer: *The combinatorial complexity of layout problems*, in: B. Preas, M. Lorenzetti (eds.), *Physical Design Automation of VLSI Systems*, pages 461–497, The Benjamin/Cummings Publishing Company, Menlo Park, 1988.

[Len90] T. Lengauer: *Combinatorial Algorithms for Integrated Circuit Layout*, Applicable Theory in Computer Science, B.G. Teubner, Stuttgart, 1990.

[Lie06] J. Lienig: *Layoutsynthese elektronischer Schaltungen*, Springer, 2006.

[LK03] C. Li, C.-K. Koh: *On improving recursive bipartitioning-based placement*, Technical Report TR-ECE 03-14, Purdue University, December 2003.

[LKS02] J. Lou, S. Krishnamoorthy, H. S. Sheng: *Estimating routing congestion using probablistic analysis*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 21(1), pages 32–41, January 2002.

[LLLC96] J. Li, J. Lillis, L.-T. Liu, C.-K. Cheng: *New spectral linear placement and clustering approach*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 88–93, 1996.

[LM90] T. Lengauer, R. Müller: *A robust framework for hierarchical floorplanning with integrated global wiring*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 148–151, 1990.

[LO73] M. C. V. Lier, R. H. J. M. Otten: *Planarization by transformation*, IEEE Transactions on Circuits and Systems CAS, volume 20(2), pages 169–171, March 1973.

[LR71] B. Landman, R. Russo: *On a pin versus block relationship for partitions of logic graphs*, IEEE Transactions on Computers, volume C-20, pages 1469–1479, December 1971.

[LRAP07] T. Luo, H. Ren, C. J. Alpert, D. Z. Pan: *Computational geometry based placement migration*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 41–47, 2007.

[LWH03] Z. Li, W. Wu, X. Hong: *Congestion driven incremental placement algorithm for standard cell layout*, in: *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 723–728, 2003.

[LXK$^+$04] C. Li, M. Xie, C.-K. Koh, J. Cong, P. H. Madden: *Routability-driven placement and white space allocation*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 394–401, 2004.

[LXK$^+$07] C. Li, M. Xie, C.-K. Koh, J. Cong, P. H. Madden: *Routability-driven placement and white space allocation*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 26(5), pages 858–871, May 2007.

[MFNK95] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani: *Rectangle-packing-based module placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1995.

[MFNK96] H. Murata, K. Fujiyoshi, S. Nakatake, Kajitani: *VLSI module placement based on rectangle-packing by the sequence-pair*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 15(12), pages 1518–1524, 1996.

[MG88] S. Mallela, L. Grover: *Clustering based simulated annealing for standard cell placement*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 25, pages 312–317, 1988.

[ML90] S. Mayrhofer, U. Lauther: *Congestion-driven placement using a new multi-partitioning heuristic*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 332–335, 1990.

[Moo65] G. E. Moore: *Cramming more components onto integrated circuits*, Electronics, volume 38(8), April 1965.

[Mül90] R. Müller: *Hierarchisches Floorplanning mit integrierter globaler Verdrahtung*, Ph.D. thesis, Universität GH Paderborn, Paderborn, 1990.

[NARM06] A. N. Ng, R. Aggarwal, V. Rachmandran, I. L. Markov: *Solving hard instances of floorplacement*, in: *International Symposium on Physical Design (ISPD)*, pages 170–177, 2006.

[NAV$^+$05] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, M. Yildiz: *The ISPD2005 placement contest and benchmark suite*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 216–219, May 2005.

[NDS01] W. Naylor, R. Donelly, L. Sha: *Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer*, U.S. Patent 6301693, October 2001.

[NFMK96] S. Nakatake, K. Fujiyoshi, H. Murata, Y. Kajitani: *Module placement on BSG-structure and IC layout applications*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 484–493, 1996.

[NRA$^+$06] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, A. B. Kahng: *A fast hierarchical quadratic placement algorithm*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 25(4), pages 678–691, April 2006.

[NSS85] S. Nahar, S. Sahni, E. Shragowitz: *Experiments with simulated annealing*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 22, pages 748–752, 1985.

[Nyq28] H. Nyquist: *Certain topics in telegraph transmission theory*, Transactions of the American Institute of Electrical Engineers, volume 47, pages 617–644, 1928.

[Obe05] B. Obermeier: *Mehrzieloptimierung beim Plazieren integrierter Schaltungen*, Ph.D. thesis, Technischne Universität München, March 2005.

[OJ04a] B. Obermeier, F. M. Johannes: *Quadratic placement using an improved timing model*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 705–710, San Diego, June 2004.

[OJ04b] B. Obermeier, F. M. Johannes: *Temperature-aware global placement*, in: *Asia and South Pacific Design Automation Conference,* volume 1, pages 143–148, Yokohama, Japan, January 2004.

[Ott82a] R. Otten: *Automatic floorplan design*, in: *ACM/IEEE Design Automation Conference (DAC),* volume 19, pages 261–267, 1982.

[Ott82b] R. Otten: *Eigensolutions in top-down layout design*, in: *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1017–1020, 1982.

[Ott83] R. Otten: *Efficient floorplan optimization*, in: *IEEE International Conference on Computer Design (ICCD)*, pages 499–501, October 1983.

[OvG89] R. Otten, L. van Ginneken: *The Annealing Algorithm*, Kluwer Academic Publishers, Dordrecht, 1989.

[PBS98] P. N. Parakh, R. B. Brown, K. A. Sakallah: *Congestion driven quadratic placement,* in: *ACM/IEEE Design Automation Conference (DAC),* pages 275–278, 1998.

[PC06] M. Pan, C. Chu: *Fastroute: A step to integrate global routing into placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* pages 464–471, 2006.

[Pri57] R. Prim: *Shortest connection networks and some generalizations*, The Bell Systems Technical Journal, pages 1389–1401, November 1957.

[PVC05] M. Pan, N. Viswanathan, C. Chu: *An efficient and effective detailed placement algorithm*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* pages 48–55, 2005.

[QB79] N. Quinn, M. Breuer: *A force directed component placement procedure for printed circuit boards*, IEEE Transactions on Circuits and Systems CAS, volume CAS-26(6), pages 377–388, June 1979.

[Qui75] N. Quinn: *The placement problem as viewed from the physics of classical mechanics*, in: *ACM/IEEE Design Automation Conference (DAC),* volume 12, pages 173–178, 1975.

[RAPM06] J. A. Roy, S. N. Adya, D. A. Papa, I. L. Markov: *Min-cut floorplacement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 25(7), pages 1313–1326, 2006.

[RC06] S. Reda, A. Chowdhary: *Effective linear programming based placement methods*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 186–191, 2006.

[RLM06] J. A. Roy, J. F. Lu, I. L. Markov: *Seeing the forest and the trees: Steiner wirelength optimization in placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 78–85, 2006.

[RM07] J. A. Roy, I. L. Markov: *Seeing the forest and the trees: Steiner wirelength optimization in placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 26(4), pages 632–644, April 2007.

[Rob83] P. Robinson: *Automatic layout for gate arrays with one layer of metal*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 20, pages 658–664, 1983.

[RPA$^+$05] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, I. L. Markov: *Capo: Robust and scalable open-source min-cut floorplacer*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 224–226, 2005.

[RPA$^+$07] H. Ren, D. Z. Pan, C. J. Alpert, G.-J. Nam, P. Villarrubia: *Hippocrates: First-do-no-harm detailed placement*, in: *Asia and South Pacific Design Automation Conference*, pages 141–146, 2007.

[RPAV05] H. Ren, D. Z. Pan, C. J. Alpert, P. Villarrubia: *Diffusion-based placement migration*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 515–520, 2005.

[RR96] M. Rebaudengo, M. S. Reorda: *Gallo: A genetic algorithm for floorplan area optimization*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 15(8), pages 943–951, 1996.

[RSV85] F. Romeo, A. Sangiovanni-Vincentelli: *Probabilistic hill climbing algorithms: Properties and application*, in: *Proceedings of the Chapel Hill Conference on VLSI*, pages 393–417, 1985.

[Saa93] Y. Saab: *Post-analysis-based clustering dramatically improves the Fiduccia-Mattheyses algorithm*, in: *European Design Automation Conference with EURO-VHDL (EURO-DAC)*, pages 22–27, 1993.

[San89] L. A. Sanchis: *Multi-way network partitioning*, IEEE Transactions on Computers, volume C-38(1), pages 62–81, January 1989.

[SB80] S. Sahni, A. Bhatt: *The complexity of design automation problems*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 17, pages 402–411, 1980.

[SB87] L. Sha, T. Blank: *ATLAS - A technique for layout using analytic shapes*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 84–87, November 1987.

[SC88] C. Sechen, D. Chen: *An improved objective function for mincut circuit partitioning*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 502–505, Santa Clara, 1988.

[Sca71] F. T. Scanlon: *Automated placement of multi-terminal components*, in: *ACM/IEEE Design Automation Workshop*, volume 8, pages 143–154, 1971.

[Sch76] D. Schweikert: *A 2-dimensional placement algorithm for the layout of electrical circuits*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 13, pages 408–416, 1976.

[SD85] L. Sha, R. Dutton: *An analytical algorithm for placement of arbitrarily sized rectangular blocks*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 22, pages 602–607, 1985.

[SDJ91] G. Sigl, K. Doll, F. M. Johannes: *Analytical placement: A linear or a quadratic objective function?*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 427–432, San Francisco, 1991.

[Sec88] C. Sechen: *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Boston, MA, 1988.

[SEM] *International technology roadmap for semiconductors*, `http://public.itrs.net`.

[Ser81] M. Servit: *Heuristic algorithms for rectilinear steiner trees*, Digital Processes, volume 7, pages 21–32, 1981.

[SH80] H. Shiraishi, F. Hirose: *Efficient placement and routing techniques for master slice lsi*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 17, pages 458–464, 1980.

[SH86] M. Shing, T. Hu: *Computational complexity of layout problems*, in: T. Ohtsuki (ed.), *Advances in CAD for VLSI*, Volume 4 (Layout Design and Verification), pages 267–294, North-Holland, Amsterdam, 1986.

[Shu75] C. F. Shupe: *Automatic component placement in the NOMAD system*, in: *ACM/IEEE Design Automation Workshop*, volume 12, pages 162–172, 1975.

[Sig92] G. Sigl: *Plazierung der Zellen bei der Layoutsynthese mittels Partitionierung und quadratischer Optimierung*, Ph.D. thesis, München, 1992.

[SJ06] P. Spindler, F. M. Johannes: *Fast and robust quadratic placement based on an accurate linear net model*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 179–186, 2006.

[SJ07a] P. Spindler, F. M. Johannes: *Fast and accurate routing demand estimation for efficient routability-driven placement*, in: *Design, Automation and Test in Europe (DATE)*, pages 1226–1231, April 2007.

[SJ07b] P. Spindler, F. M. Johannes: *Kraftwerk — a fast and robust quadratic placer using an exact linear net model*, in: G.-J. Nam, J. Cong (eds.), *Modern Circuit Placement — Best Practices and Results*, chapter 4, pages 59–91, Springer, 978th edition, September 2007.

[SK72] D. Schweikert, B. Kernighan: *A proper model for the partitioning of electrical circuits*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 9, pages 57–62, 1972.

[SK87] P. R. Suaris, G. Kedem: *Quadrisection: A new approach to standard cell layout*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 474–477, November 1987.

[SK88] P. Suaris, G. Kedem: *An algorithm for quadrisection and its application to standard cell placement*, IEEE Transactions on Circuits and Systems CAS, volume CAS-35(3), pages 294–303, March 1988.

[SKAS88] T. Shiple, P. Kollaritsch, J. Allen, D. Smith: *Area evaluation metrics for transistor placement*, in: *IEEE International Conference on Computer Design (ICCD)*, 1988.

[SKK$^+$93] H. Shin, C. Kim, W. Kim, M. Oh, K. Rhee, S. Choi, H. Chung: *A combined hierarchical placement algorithm*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 164–169, 1993.

[SM90] K. Shahookar, P. Mazumder: *A genetic approach to standard cell placement using meta-genetic parameter optimization*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-9(5), pages 500–511, May 1990.

[SS91] G. Sigl, U. Schlichtmann: *Goal oriented slicing enumeration through shape function clipping*, in: *European Conference on Design Automation (EDAC)*, volume 2, pages 361–365, Amsterdam, 1991.

[SS93] W.-J. Sun, C. Sechen: *Efficient and effective placement for very large circuits*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 170–177, 1993.

[SS95] W.-J. Sun, C. Sechen: *Efficient and effective placement for very large circuits*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 14(3), pages 349–359, March 1995.

[SSJ08a] P. Spindler, U. Schlichtmann, F. M. Johannes: *Abacus: Fast legalization of standard cell circuits with minimal movement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 47–53, April 2008.

[SSJ08b] P. Spindler, U. Schlichtmann, F. M. Johannes: *Kraftwerk2 - a fast force-directed quadratic placement approach using an accurate net model*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, 2008, to appear.

[SSL93] S. Sutanthavibul, E. Shragowitz, R. Lin: *An adaptive timing-driven placement for high performance vlsi's*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 12(10), pages 1488–1498, 1993.

[SSV85] C. Sechen, A. Sangiovanni-Vincentelli: *The TimberWolf placement and routing package*, IEEE Journal of Solid-State Circuits SC, volume SC-20(2), pages 510–522, April 1985.

[SSV86] C. Sechen, A. Sangiovanni-Vincentelli: *TimberWolf3.2: A new standard cell placement and global routing package*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 23, pages 432–439, 1986.

[SU72] D. Schuler, E. Ulrich: *Clustering and linear placement*, in: *ACM/IEEE Design Automation Workshop*, volume 9, pages 50–56, 1972.

[SW97] M. Sarrafzadeh, M. Wang: *NRG: Global and detailed placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 532–537, 1997.

[SY95] Sait, Youssef: *VLSI PHYSICAL DESIGN AUTOMATION Theory and Practice*, McGRAW-HILL Book Company Europe, 1995.

[SZJ06] M. Saedi, M. S. Zamani, A. Jahanian: *Prediction and reduction of routing congestion*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 72–77, 2006.

[TKH88a] R.-S. Tsay, E. S. Kuh, C.-P. Hsu: *Module placement for large chips based on sparse linear equations*, International Journal of Circuit Theory and Applications, volume 16, pages 411–423, 1988.

[TKH88b] R.-S. Tsay, E. S. Kuh, C.-P. Hsu: *PROUD: A fast sea-of-gates placement algorithm*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 25, pages 318–323, 1988.

[TKH88c] R.-S. Tsay, E. S. Kuh, C.-P. Hsu: *PROUD: A sea-of-gates placement algorithm*, IEEE Design & Test of Computers, pages 44–56, December 1988.

[TYC05] T. Taghavi, X. Yang, B.-K. Choi: *Dragon2005: Large-scale mixed-size placement tool*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 245–247, 2005.

[VC05] N. Viswanathan, C. C.-N. Chu: *Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 24(5), pages 722–733, May 2005.

[vG90] L. P. P. P. van Ginneken: *Buffer placement in distributed rc-tree networks for minimal elmore delay*, in: *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 865–868, 1990.

[Vij89] G. Vijayan: *Min-cost partitioning on a tree structure and applications*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 26, pages 771–774, Las Vegas, 1989.

[VK05a] K. Vorwerk, A. Kennings: *An improved multi-level framework for force-directed placement*, in: *Design, Automation and Test in Europe (DATE)*, pages 902– 907, 2005.

[VK05b] K. Vorwerk, A. Kennings: *Mixed-size placement via line search*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 899–904, 2005.

[VKV04] K. Vorwerk, A. Kennings, A. Vannelli: *Engineering details of a stable force-directed placer*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 573– 580, 2004.

[vLA87] P. van Laarhoven, E. Aarts: *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht/Boston/Lancaster/To, 1987.

[VNA$^+$07] N. Viswanathan, G.-J. Nam, C. J. Alpert, P. Villarrubia, H. Ren, C. Chu: *RQL: Global placement via relaxed quadratic spreading and linearization*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 453–458, 2007.

[VPC06] N. Viswanathan, M. Pan, C. Chu: *Fastplace 2.0: An efficient analytical placer for mixed-mode designs*, in: *Asia and South Pacific Design Automation Conference*, pages 195–200, 2006.

[VPC07] N. Viswanathan, M. Pan, C. Chu: *Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control*, in: *Asia and South Pacific Design Automation Conference*, pages 135–140, 2007.

[Vyg97] J. Vygen: *Algorithms for large-scale flat placement*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 746–751, 1997.

[Waw88] K. Wawryn: *Layout including parasitics for printed circuit boards*, International Journal of Circuit Theory and Applications, volume 16, pages 107–128, 1988.

[WBG04] J. Westra, C. Bartels, P. Groeneveld: *Probabilistic congestion prediction*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 204–290, 2004.

[WC04] M.-C. Wu, Y.-W. Chang: *Placement with alignment and performance constraints using the b*-tree representation*, in: *IEEE International Conference on Computer Design (ICCD)*, pages 568–571, 2004.

[Whi84] S. White: *Concepts of scale in simulated annealing*, in: *IEEE International Conference on Computer Design (ICCD)*, pages 646–651, 1984.

[Wil80] W. E. Williams: *Partial Differential Equations*, Oxford University Press, 1980.

[Wip85] G. J. Wipfler: *Ein Plazierungskonzept für den Bottom-up-Entwurf hochintegrierter Schaltungen*, number 57 in Fortschrittsberichte VDI Reihe 9: Elektrotechnik/E, VDI Verlag, Düsseldorf, 1985.

[WL86] D. Wong, C. Liu: *A new algorithm for floorplan design*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 23, pages 101–107, 1986.

[WLL88] D. Wong, H. Leong, C. Liu: *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, Boston/Lancaster/Dordrecht, 1988.

[WM87] B. X. Weis, D. A. Mlynski: *A new relative placement procedure based on MSST and linear programming*, in: *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 564–567, 1987.

[WM88] B. X. Weis, D. A. Mlynski: *A graphtheoretic approach to the relative placement problem*, IEEE Transactions on Circuits and Systems CAS, volume CAS-35(3), pages 286–293, March 1988.

[WS99] M. Wang, M. Sarrafzadeh: *On the behaviour of congestion minimization during placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 145–150, 1999.

[WS00] M. Wang, M. Sarrafzadeh: *Modeling and minimization of routing congestion*, in: *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 185–290, 2000.

[WWM82] G. J. Wipfler, W. Wiesel, D. A. Mlynski: *A combined force and cut algorithm for hierarchical VLSI layout*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 19, pages 671–676, 1982.

[WYES00] M. Wang, X. Yang, K. Eguro, M. Sarrafzadeh: *Multi-center congestion estimation and minimization during placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 147–152, 2000.

[WYS00] M. Wang, X. Yang, M. Sarrafzadeh: *Congestion minimization during placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 19(10), pages 1140–1148, October 2000.

[XMFR04] Z. Xiu, J. D. Ma, S. M. Fowler, R. A. Rutenbar: *Large-scale placement by gridwarping*, in: *ACM/IEEE Design Automation Conference (DAC)*, pages 351–356, 2004.

[XR07] Z. Xiu, R. Rutenbar: *Mixed-size placement with fixed macrocells using grid-warping*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 103–109, 2007.

[YCS02] X. Yang, B.-K. Choi, M. Sarrafzadeh: *Routability-driven white space allocation for fixed-die standard-cell placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 42–49, 2002.

[YCS03] X. Yang, B.-K. Choi, M. Sarrafzadeh: *Routability-driven white space allocation for fixed-die standard-cell placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 22(4), pages 410–419, April 2003.

[YK92] Y. Y. Yang, C. M. Kyung: *HALO: An efficient global placement strategy for standard cells*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume CAD-11(8), pages 1024–1031, August 1992.

[YKM$^+$03] A. A. M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, P. H. Madden: *Fractional cut: Improved recursive bisection placement*, in: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 307–310, 2003.

[YKS01] X. Yang, R. Kastner, M. Sarrafzadeh: *Congestion estimation during top-down placement*, in: *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 164–169, 2001.

[YKS02] X. Yang, R. Kastner, M. Sarrafzadeh: *Congestion estimation during top-down placement*, IEEE Transactions on Computer-Aided Design of Circuits and Systems, volume 21(1), pages 72–80, January 2002.

[You03] S. Yousef: *Iterative Methods for Sparse Linear Systems*, Cambridge University Press, 2003.

[ZD02] K. Zhong, S. Dutt: *Algorithms for simultaneous satisfaction of multiple constraints and objective optimization in a placement flow with application to congestion control*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 39, pages 854–859, 2002.

[Zim88] G. Zimmermann: *A new area and shape function estimation technique for VLSI layouts*, in: *ACM/IEEE Design Automation Conference (DAC)*, volume 25, pages 60–65, 1988.

List of Variables

$A_{\cup}$	Area of the union of all modules	42
A_{chip}	Area of the chip	41
$A_{\mathit{mod,i}}$	Area of module i	40
$A_{\mathit{mod,tot}}$	Area of all modules	41
$A_{\mathit{net,n}}$	Area of the net rectangle of net n	62
$\mathbf{C_x}$	System matrix in x-direction	36
$\mathbf{C_y}$	System matrix in y-direction	36
$\mathbf{\Delta x}$	Change of the module positions in x-direction	42
$\mathbf{d_x}$	System vector in x-direction	36
$\mathbf{d_y}$	System vector in y-direction	36
$\mathbf{F_x^{hold}}$	Hold force in x-direction	44
$\mathbf{F_x^{move}}$	Move force in x-direction	44
$\mathbf{F_x^{net}}$	Net force in x-direction	39
$\mathring{\mathbf{C}}_{\mathbf{x}}$	Diagonal matrix with the weights of the move force	44
$\mathbf{\Phi_x}$	Gradients of the potential	44
$\mathbf{x}$	Positions of movable modules in x-direction	36
$\mathbf{y}$	Positions of movable modules in y-direction	36
$\boldsymbol{\nabla}$	Nabla operator in x-y space	50
$\boldsymbol{\nabla_x}$	Nabla operator in x-direction	37
$\boldsymbol{\nabla_y}$	Nabla operator in y-direction	37
δ	Average length of the potential's gradient	53
$(\Delta x_i, \Delta y_i)$	Change of the position of module i during a placement iteration	40
$d_{\mathit{mod,i}}$	Individual module density of module i	40
$D_{\mathit{mod}}\,(x, y)$	Module demand-and-supply system	43
$D_{\mathit{rout}}\,(x, y)$	Routing demand-and-supply system	65
d_{sup}	Module supply density	43
$D_{\mathit{mod,i}}^{\mathit{dem}}\,(x, y)$	Demand of module i	42
$D_{\mathit{mod}}^{\mathit{dem}}\,(x, y)$	Module demand	42
$D_{\mathit{rout}}^{\mathit{dem}}\,(x, y)$	Routing demand	62
$D_{\mathit{rout,n}}^{\mathit{dem}}\,(x, y)$	Demand of net n	62
$D_{\mathit{mod}}^{\mathit{sup}}\,(x, y)$	Module supply	43
$D_{\mathit{rout}}^{\mathit{sup}}\,(x, y)$	Routing supply	64
$d_{\mathit{wire,n}}$	Wire density of net n	62
$D(x, y)$	Generic demand-and-supply system	42
e_i	Weight of module i	70

$e = (p, q)$	One two-pin connection	29
F	Numbers of fixed modules	37
$\frac{\partial}{\partial x_i}\Gamma_{x,pq}$	Derivative of the quadratic cost function of a two-pin connection	38
Γ	Quadratic cost function	29
$\Gamma_{n,x}$	Quadratic cost function of net n in x-direction	30
Γ_n^{HPWL}	Half-perimeter wirelength of net n	30
Γ_x	Quadratic cost function in x-direction	37
$\Gamma_{x,pq}$	Quadratic cost function of a two-pin connection	37
$\left(x_p^{pin}, y_p^{pin}\right)$	Position of pin p	29
L_n	Wirelength of net n	62
$\mathring{w}_i$	Weight of the move force of module i	48
$\mathring{x}_i$	Target point of module i	44
$\mathcal{M}$	Set of modules	29
$\mathcal{N}$	Set of nets	29
$\mathcal{P}$	Set of pins	29
M	Number of movable modules	36
μ	Average module movement	41
μ_D	Average of the demand-and-supply system	52
Ω	Module overlap	41
Φ	Poisson potential	43
π	Function to map set $\mathcal{P}$ of pins to the set $\mathcal{M}$ of modules	36
R	Rectangle function	41
σ_D	Standard deviation of the demand-and-supply system	52
td	Module target density	57
w_i, h_i	Width and height of module i	40
w_{chip}, h_{chip}	Width and height of the chip	41
$w_{net,n}, h_{net,n}$	Width and height of the net rectangle of net n	62
(x_i, y_i)	Position of module i after legalization	70
(x_i, y_i)	Position of module i at the end of a placement iteration	40
(x'_i, y'_i)	Position of module i at the start of a placement iteration	40
(x'_i, y'_i)	Position of module i before legalization	70
(x_{chip}, y_{chip})	Position of the lower left corner of the chip	41
$(x_{net,n}, y_{net,n})$	Position of the net rectangle of net n	62
(x_p^{off}, y_p^{off})	Offset of pin p	36

List of Figures