Placement Techniques for VLSI Layout Using Sequence-Pair Legalization

Jens Egeblad

Master of Science Thesis
Department of Computer Science
University of Copenhagen

July 1st, 2003

Contents

Abstract

This thesis considers the placement problem in VLSI layout, which deals with layout optimization of integrated circuits. Most practical formulations of this problem are NP-hard. The most widely used formulations of this problem consider placement of rectangles within a rectangular container. The rectangles represent logical components and are called modules. Modules are connected by wires and a common objective is to minimize the wire-length.

Most authors consider modules of equal height, the so-called standard-cells, but often practical instances of the problem also contain rectangles of varied height. Therefore there is a need for placement methods which are not limited to standard-cell placement. These problems fall in two groups mixed-cell placement which is a combination of standard-cells and a limited number of larger modules of unequal height, and general-cell placement which contain modules of arbitrary sizes.

The placement process is commonly broken in two parts; global and final placement. Global placement generates a good initial solution which often contains overlap. Final placement generally optimizes each module of the placement individually and generates non-overlapping legal solutions.

We define the problem with different wire-length objective functions and present a survey of successful placement techniques from the last decade. We present a novel legalization technique for general-cell instances which has asymptotic running time $O(n \log n)$, and is capable of removing overlap from placements which is a well-known problem for many placement heuristics.

We use the legalization algorithm to create a new global placement method which combines legalization and well-known analytic methods.

We present a new final placement method which is based on relaxation based neighborhood search and use the legalization algorithm to remove overlap during optimization. Experimental results are reported for the global and final placement heuristics on standard- mixed- and general-cell circuits.

Experiments show promising results. New general-cell circuits based on common standard-cell circuits are placed with netlengths which are less than 15% worse than their standard-cell counterparts. Also the new placement algorithm produces results comparable and in some cases better than those previously published and produced by commercial placement tools.

Preface

This is my M. Sc. thesis written in the period from July 2002 to July 2003 at the Department of Computer Science, DIKU, University Copenhagen. Many thanks goes to my counsellor Professor David Pisinger who listened patiently to many ideas and suggested many improvements. I would also like to thank Associate Professor Martin Zachariasen for providing me with the real-life IBM-circuits. Thanks also goes to Hans Eisenmann at PDF Solutions, Inc. for fast response to my e-mail queries. Finally I would like to thank Frank Johannes at the Technische Universität München for providing me with the paper on the Domino placer and Jens Vygen at the Research Institute for Discrete Mathematics University of Bonn for providing me with a currently unpublished paper on legalization.

About the Reader

The VLSI-layout problem is complicated and not just in a combinatorial sense. However, although not a main goal of this thesis, I have tried to make the text readable even by novices in the field of VLSI-layout. I do expect the reader to be familiar with certain fundamental concepts of data-structures, algorithms and combinatorial optimization. These include

- Graphs, search trees, algorithms and running time, minimum spanning trees, steiner minimal trees, NP-hard problems.
- Multi-variable calculus and algebra.
- Linear programming and combinatorial optimization.
- Local search, simulated annealing and other meta-heuristics.

However a reader unfamiliar with some of these concepts may still understand the majority of the text.

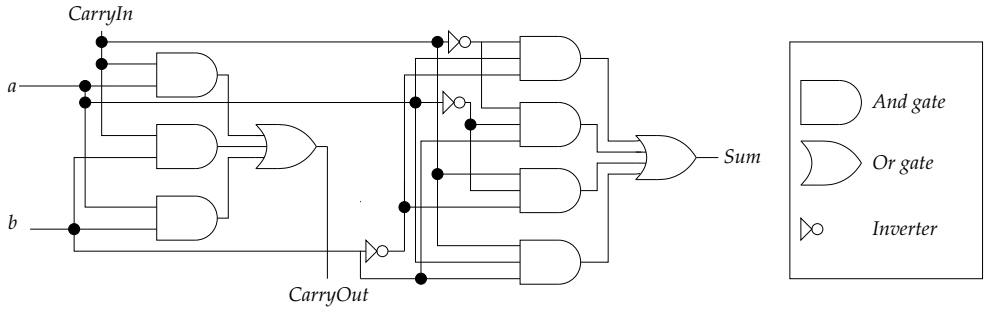


Figure 1.1: *Simple example of logic. This is a full one bit-adder.* a*,* b *and* $CarryIn$ *are input signals.* $CarryOut$ *and* Sum *are output signals.*

1 Introduction

The purpose of this thesis is to consider novel techniques for layout optimization of integrated circuits. Before we describe the problem in detail we will give a brief introduction to the manufacturing process. This will be followed by an elaboration of the layout problem and an overview of the subsequent chapters.

1.1 Design and Construction of an Integrated Circuit

When the overall design goals of an IC has been decided the manufacturing process of it can roughly be divided into the following parts.

- **Architectural and logic design** Based on the design goals the architecture of the IC is established and the architectural description is converted into logic components (gates) of the form `or`, `and`, `not` etc., which are connected appropriately to each other (see figure 1.1). Note that the logic components can also be a combination of many connected components, such as a 32-bit adder or an ALU (arithmetic logical unit).
- **Circuit layout** Based on the logical design a physical design description is created. The logical components are now represented as modules which must be positioned appropriately within a rectangular region. This phase is now commonly done by computers and part of it is the focus of this thesis.
- **Physical construction** When the layout phase has been completed actual construction of the circuit can commence. The layout is transfered to a piece of silicon.

Only the circuit layout phase is considered in this thesis, however in order to understand it we need to consider the physical construction in more detail.

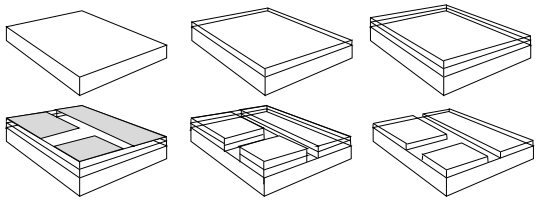


Figure 1.2: *Growing an integrated circuit on a wafer. First the the silicon dioxide is added. Then a photo resist chemical is added. The photo resist is exposed to ultraviolet light emitted through a mask. Chemicals remove the photo resist and the underlying silicon dioxide. The result is a pattern on the wafer which can become conducting or insulating through further chemicals processes.*

1.1.1 Physical Construction

An integrated circuit is constructed by using the semi-conducting properties of silicon (Si - Element 14 in the periodic table). Silicon by itself is a mediocre conductor, however silicon's conducting capabilities can be altered by chemicals to make it either a fine conductor or insulator. This observation is the foundation of integrated circuits.

For production of ICs, silicon is delivered in cylinders called ingots which have diameters in the order of decimeters. According to Intel's home-page [1] they currently use 30 cm ingots for the Pentium IV production. The ingots are cut into thin slices only a few millimeters thick. The slices are called wafers. The IC is placed on a wafer using a technology called lithography. The process consists of many similar steps of the kind:

- Grow silicon dioxide.
- Add photo resist chemical.
- Emit ultraviolet light through a mask and expose *parts* of the photo resist layer.
- Remove exposed areas of photo resist and underlying silicon dioxide with chemicals.
- Make remaining silicon dioxide conducting or insulating by further chemical steps.

The process is shown on figure 1.2 and can be repeated to grow a number of layers on top of each other on the chip. The Pentium 4 chip uses approximately 20 layers. Layers can be connected and connections between layers are called *vias*. Commonly the lower layers consists of logical components while upper layers contain wires which connect the components; so called *routing layers*.

Usually a wafer contains many ICs and each area containing an IC is called a die (plural dice). The silicon ingots often contain a number of anomalies which would

[1] www.intel.com

Year	IC	Transistors	Feature size (micron)	Die size (mm^2)
1982	Intel 286	134.000	-	-
1989	Intel 486	1.200.000	1	79
1993	Intel Pentium	3.100.000	0.5	161
1997	Motorola Power PC G3	6.350.000	0.22	-
1999	Motorola Power PC G4	10.500.000	0.20	-
2000	AMD Athlon (original)	22.000.000	0.25	184
1999	Intel Pentium III	28.000.000	0.18	106
2001	Intel Pentium 4	42.000.000	0.13	116

Table 1.1: *A list of popular integrated circuits and the number of transistors on each circuit.*

generate defects in the ICs. Making each IC as small as possible will increase the number of ICs on a wafer (also called *yield*) and thereby decrease the number of circuits affected by the anomalies.

The logic of the IC is achieved with transistors that are connected with wires. The number of transistors and the die size have increased steadily during past years. Table 1.1 shows this development.

1.2 The Layout Description

As the previous paragraph explained the logic of the chip must be converted to a mask that can be used to create the transistors and wires of the IC. A simple description of the circuit is a hypergraph in which each node corresponds to a logic component or even a transistor.

In some cases a hierarchical formulation is used. Here the logical components are embedded into larger components which in turn may be part of even larger components. The purpose of the hierarchical design methodology is clear; to reduce data and problem complexity.

To hide the internal complexity during the layout phase, components are most often considered black box rectangular elements with pins. Different components are connected by attaching wires to the pins.

In this text we call the black-box components *modules* although other writers use different terms; cell, circuit, macro or block. The terms signify the complexity of the underlying components. E.g. cells are very simple structures like `and` and `or` gates, blocks consists of many cells. We use the term module for the general case but may in some cases use the more precise terms.

Modules could have any shape but they are commonly described by rectangles, although some writes consider *rectilinear* modules (irregular polygons with only hori-

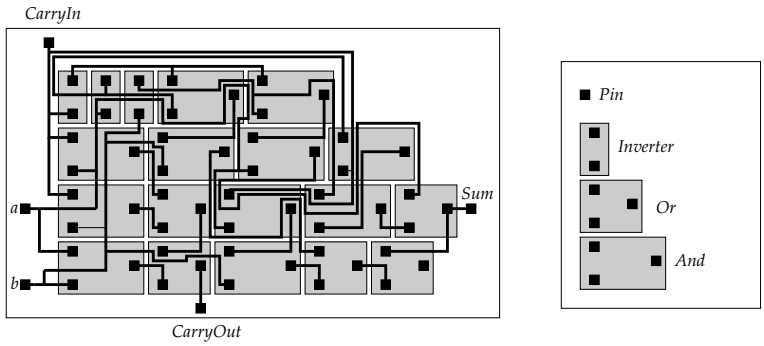


Figure 1.3: *A* very *simple example of a circuit layout consisting of* `inverters`, `and` *and* `or` *gates. The logic of the circuit is the adder of figure 1.1. The large hollow rectangle is the real estate. Modules are shown as dark shaded rectangles and the legend illustrates the three types of modules. IO-pins are shown as pins which are not placed on modules. Finally wires are shown as lines. Vias have been left out of the figure. Also one routing layer is sufficient since no wires cross.*

zontal and vertical edges).

The area on which the modules are to be placed is of course the die but in this context more commonly referred to as the *real-estate*.

Complete circuits also contain IO-pins (or pads) which connect the circuit to the "outside world". These are often aligned along the borders but in some cases they are distributed all over the real-estate. When aligned on the border the IO-pins could have either predetermined positions or be allowed any position on the border.

1.3 The Layout Problem

The layout problem is to convert the circuit description to a physical layout which can be used to create masks for the circuit construction.

Figure 1.3 is a *very* simple example of the logic from 1.1 converted to a physical layout. The figure deserves a few comments. First of all it should be noted that the adder shown on the figure is not a complete circuit. Among the things missing is the clock signal. It consists solely of `and`, `or` and `inverter` gates which are shown as shaded modules. The actual physical description of the three types of modules has been completely hidden. The figure also illustrates IO-pins of the circuit which are black rectangles not placed on any modules. Wires are also shown. It should be noted that normally wires are routed on at least two different layers; One for horizontal and one for vertical wires in which case one would require a connection between the two layers (*via*) at every corner of a wire. We have omitted the vias to keep the figure simple and also one routing layer suffices since no two wires cross.

Because of its complexity the layout problem is usually divided into two major subproblems; placement and routing. Placement deals with placing the modules on the

real-estate. Routing deals with positioning wires between the modules. The division is not strict. Often some form of rough estimate of wire-length is considered during placement

1.3.1 The Placement Problem

The placement problem – which is the focus of this thesis – is to position the modules on the real-estate such that some form of objective function is minimized. The objective function is often one or several of the following: minimize expected length of longest wire, minimize expected sum of wire-length, minimize congestion of wires or minimize area.

The purpose of minimizing wire-length is to reduce signal delay which in turn may increase possible clock frequency. Internally modules also contribute to signal delay but we will not discuss this here. Minimizing area is used as an objective for at least two reasons: in the hope that it will reduce netlength and to increase yield. Congestion may simplify routing, reduce signal interference and distribute heat emission evenly on the real estate. The objectives will be discussed in more detail in section 2.2.6.

Because the wire-lengths are connected to routing, placement algorithms often use an estimate for wire-length during the placement optimization. Wire-length estimates will be discussed in section 2.3

The final circuit may be confined to a specific layout style which describes shape and position of the modules. The layout styles vary from semi-custom layout where the design is restricted, to full custom layout where the designer has maximal freedom. There exists five major layout styles:

1. **Gate array** The gate array design style consists of a prefabricated silicon with identical modules distributed evenly on the real-estate. The function of a module is determined solely by its connections. Therefore the entire logic is determined by the wires. Space has been reserved for routing which occurs between cells.

2. **Sea-of-gates** The sea-of-gates layout is similar to gate array but no space is reserved for routing. Instead the entire real estate has been filled with prefabricated transistors. Some of the transistors become unusable however since space must still be allocated for routing.

3. **Standard-cell** The standard-cell layout is by far the most popular of the literature on the placement problem. In this case modules have identical height but varies in width. Modules are placed in rows with space between them. The modules may be taken from a library of predefined basic modules. Originally routing was done between rows but multilayer technology now allows for routing anywhere on the real-estate.

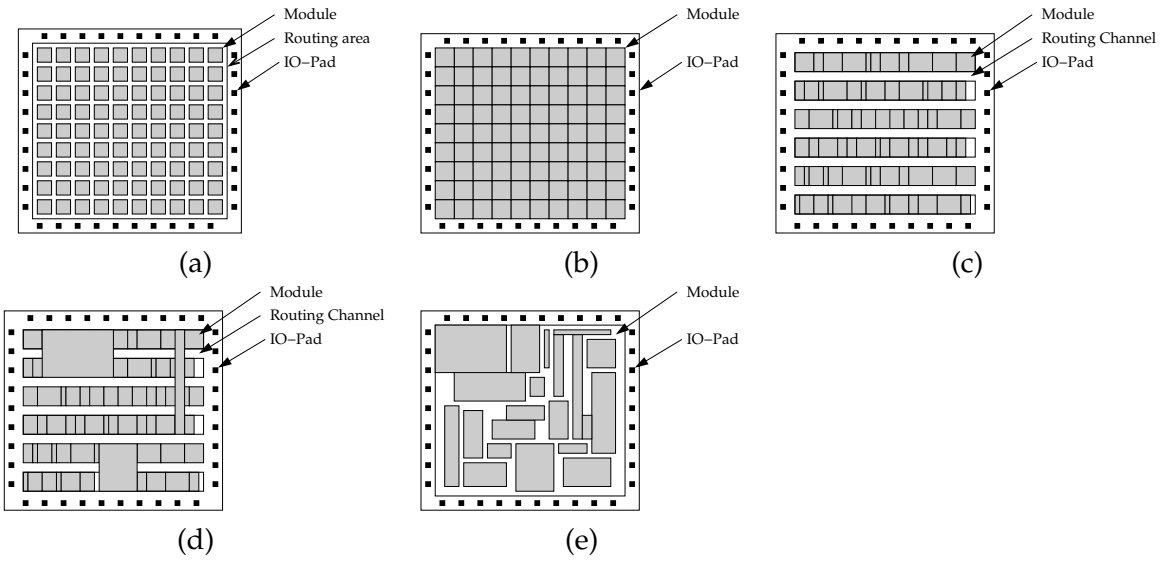


Figure 1.4: *The different major layout styles. (a) Gate array. (b) Sea-of-gates. (c) Standard-cells. (d) Mixed cell layout. (e) general-cell layout.*

4. **Mixed-cell** The mixed-cell model is similar to standard-cell layout but allows for large modules in the layout which may vary in height and width and cover several rows.

5. **General-cell (Macros)** The final layout style which is also the *only full-custom* is the general-cell layout style. In this case modules are allowed any size and position on the real estate. In some cases modules are even allowed rectilinear shapes .

As mentioned only the general-cell layout style is characterized as full-custom layout. The other types of layout are characterized as semi-custom. In this thesis we will only consider standard-cell, mixed-cell and general-cell layout and we will only consider rectangular modules. Figure 1.4 illustrates the different layout styles.

Placement methods are commonly divided into two groups; global and detailed (final) placement. Global gives a rough estimate or relative position of modules. Final placement finalizes the placement by local optimization.

1.3.2 Routing

After placement the position of the wires is determined. This is called routing. The routing solutions are restricted by the solution of the placement step; we cannot move modules at this point. Therefore the quality of a placement depends on how well it can be routed. The routing problem is often solved in two steps; global routing

and detailed routing. Global routing determines how wires are placed relative to the modules. Detailed routing determines exact position of the wires.

1.3.3 Pre- and Post-processing

In most cases there are two more steps; preprocessing and post-processing. Preprocessing may divide the circuit into sub-circuits which are optimized individually but can be combined during the complete circuit optimization phase. Post-processing may compact and uncompact the circuit. Compaction will reduce the necessary area whereas uncompaction will distribute modules and wires to reduce signal interference or even out heat distribution.

1.3.4 The Placement and Routing Problems are NP-hard

Although it depends on the exact formulation most practical versions of the placement problem are NP-hard (see e.g. [54]). In section 2.2.6 we will give a proof that placement based on standard-cell or general-cell models is NP-hard independent of the objective function.

Routing is also in general NP-hard. We point to [54] for proofs of this claim.

1.4 Motivation and Objective of this Thesis

Although the layout problem has existed since the invention of the integrated circuit local search methods of recent years have shown that placements can still be improved. One problem with some local search methods is to deal with overlap. Also with ICs reaching 100.000.000 transistors[2] efficient methods capable of handling large circuits are necessary. Little attention has been given to mixed- and general-cell layout. Mixed-cell layout is generally handled by considering special cases for macros and full-custom layout is generally avoided because of its increased complexity. Also with more transistors and increased functionality of the circuits there is growing need for the ability to split the circuit components into modules which can be optimized separately and later combined. These modules would likely have arbitrary size.

The purpose of this thesis is to investigate a legalization technique for solving the placement algorithm. The technique should allow us to remove overlap from a *general-cell* placement. We will use the technique in connection with both a global- and final placement and for both standard-cell and general-cell layout styles. The legalization algorithm is based on a well-known abstract representation of rectangle placements; sequence-pair, and is not limited to standard-cells such as most efficient current methods.

[2] Intel are working towards 1 billion transistors in 2007 [73]

Objective function and simplifications As objective function we have decided to minimize wire-length. This is the most common objective of the literature and it allows us to compare results with other authors. To handle the layout-problem some simplified assumptions are made. In general we will not consider problems such as electric interference, internal signal-delay in modules, congestion and routability. But these simplifications are common in the VLSI-placement literature. We give a more precise formulation of the layout problem in section 2.

1.5 Outline of the Thesis and our Contributions

The thesis is divided in roughly two parts. The following two sections consider formulation of the placement problem (section 2) and previous work (section 3). These two sections can be skipped by readers which are experienced with the VLSI-placement problem.

The remaining sections describe the legalization algorithm and the associated global and final placement heuristics. First, in section 4, we will present the legalization algorithm which is based on the sequence-pair representation. Then in section 5 we will introduce the benchmark circuits to give the reader a better understanding of their complexity. In section 6 we present a local search method which is an important element to both our global- and final-placement methods. The new global-placement method is presented in section 7 and the final-placement method is presented in section 8. To test our new methods we report experimental results on well-known benchmark circuits in section 9. Finally in section 10 we give our conclusion on the new placement methods and list future directions.

Some details have been omitted from the main text and can be found in the appendices. Readers who wish to experiment with their own heuristic may find inspiration in appendix A where we have listed a number of preliminary approaches that proved unsuccessful and our explanation as to why the methods did not work.

Our contributions in this thesis can be summarized as follows:

- *Extensive survey* In section 3 we give an extensive survey of previous solution methods with focus on the methods from the last ten years. To our knowledge no other recent survey is quite as extensive.
- *Extensions to the sequence-pair placement algorithms* We have extended an existing algorithm for converting a sequence-pair to a placement so that it can handle more constraints and produce more compact placements.
- *Placement-to-sequence-pair* We have developed a placement-to-sequence-pair-agoritem which can convert a placement to a sequence-pair even if the placement contains overlapping rectangles.

- *A Legalization algorithm.* The placement-to-sequence-pair algorithm and the sequence-pair-to-placement algorithm have been combined to form a legalization algorithm which can remove overlap from a placement. This and the previous two items are the focus of section 4.
- *New general-cell circuits* In section 5 we will describe current as well as propose three new benchmark circuits.
- *New Global placement method* Based on the legalization algorithm we have created a new global placement heuristic which can handle standard-cell, mixed-cell and general-cell layouts. This is the focus of section 7.
- *New final-placement* We have also developed a new final-placement method which is the focus of section 8.

Results Experiments show that the new placement heuristics developed perform well with both standard-cell and general-cell circuits. For the standard-cell circuits we are able to produce results which are comparable to previously published and commercially produced results. We are also able produce promising results for new general-cell benchmarks. These results are good considering that our new placement heuristics are only prototypes and much fine-tuning could probably improve results both with respect to time and objective value.

2 Preliminaries

In this section we will give a more formal definition of the VLSI-placement problem. Also we will introduce some well-known basics of the VLSI-placement problem which is considered common knowledge in the VLSI-placement field. These basics are crucial for the understanding of section 3 which deals with previous work.

Experienced readers may skip the section and use it solely for reference. However we do recommend skimming the text if for no other reason than to get a feel of our notation.

2.1 General Conventions

We use numbers and graphs extensively throughout the subsequent sections but use standard conventions.

2.1.1 Numbers

- $\mathbb{N}$ is the set of positive integers $\mathbb{N} = \{1, 2, 3, 4, 5, \ldots\}$.
- $\mathbb{N}_0$ is the set of non-negative integers $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.
- $\mathbb{Z}$ is the set of integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
- $\mathbb{Q}$ is the set of rational numbers $\mathbb{Q} = \{\frac{q}{p} \mid p \in \mathbb{N}, q \in \mathbb{Z}\}$.
- $\mathbb{R}$ is the set of real numbers.
- $\mathbb{R}_+$ is the set of non-negative real numbers $r \geq 0, r \in \mathbb{R}$
- $\mathbb{R}^n$ is the set of n-dimensional vectors $\mathbf{v} \in \mathbb{R}^n$.
- $\mathbb{R}^{m \times n}$ is the set of $n \times m$ matrices (n rows and m columns).

In general vectors and matrices are written in **bold** face. v_i is the ith element of vector $\mathbf{v} \in \mathbb{R}^n$ and a_{ij} is the element at the ith row and jth column of matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. For vectors $\mathbf{v} \in \mathbb{R}^2$ v_x is the x-component and v_y the y-component.

For intervals we let $I = [x_1, x_2] \subset \mathbb{R}$ be the closed interval which contains x_1 and x_2. $I = [x_1, x_2[\subset \mathbb{R}$ is the half-open interval which contains x_1 but not x_2 and $I =]x_1, x_2[\subseteq \mathbb{R}$ is the open interval which contains neither x_1 nor x_2.

2.1.2 Graphs and Hypergraphs

In general $G = (V, E)$ is a graph in which V is the set of vertices and $E \subseteq V \times V$ is the set of edges.

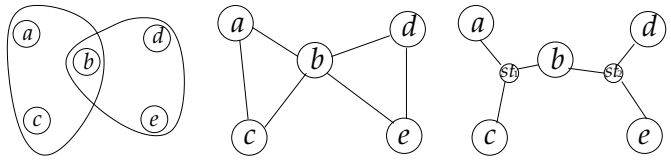


Figure 2.1: *A hypergraph is converted to a standard graph. (A) The hypergraph. The edges in the hypergraph ($\{a, b, c\}$ and $\{b, d, e\}$) are subsets and are illustrated as circumscribing curves. (B) The hyperedges have been converted to sub-cliques. (C) The hyperedges have been converted to star-graphs. One new point - the star point - is introduced for each edge.*

Hypergraphs Hypergraphs are extensions of graphs with *hyperedges*. A hyperedge e is a subset of vertices $e \subseteq V$. Edges of regular graphs are hyperedges with cardinality 2. Hypergraphs are in general referred to as $H = (V, E)$. There is no way to convert a hyperedge e to a regular edge without changing some properties of the graph. Two common approaches exists however:

1. Convert e to a clique subgraph by adding edges between every pair of vertices in the hyperedge.
2. Add one new vertex st_e, and convert e to a star subgraph by adding edges between every vertex $v \in e$ and st_e

The two methods are illustrated on figure 2.1.

2.2 A VLSI Placement Instance

The VLSI placement problem deals with nets and modules. The nets connect the modules through pins. Throughout this text we use the following conventions:

- The set of modules is $\mathcal{M}$.
- The set of pins is $\mathcal{P}$.
- The set of nets is $\mathcal{N}$.
- The placement area with IO-pins is $\mathcal{A}$.

Combined modules, pins and nets constitute a circuit $\mathcal{C} = (\mathcal{M}, \mathcal{P}, \mathcal{N}, \mathcal{A})$.

2.2.1 Modules

For module $m \in \mathcal{M}$ we define the following:

- A width $w(m)$ ($w : \mathcal{M} \to \mathbb{N}$).
- A height $h(m)$ ($w : \mathcal{M} \to \mathbb{N}$).

Some modules are blockages or have fixed position and orientation. Let $\mathcal{F}$ be the set of fixed modules and blockages (we assume all blockages are rectangular). For fixed modules we let $(x_f(m), y_f(m))^t \in \mathbb{Z} \times \mathbb{Z}$ be the position of the module. The placement area is in general rectangular and contains input/output pins of the circuit. These are also fixed. To simplify our notation the placement area $\mathcal{A}$ is also a fixed module. So we have $\mathcal{F} \subset \mathcal{M} \cup \{\mathcal{A}\}$ and $\mathcal{A} \in \mathcal{F}$. Also let $\mathcal{MA} = \mathcal{M} \cup \{\mathcal{A}\}$ be the set of modules including the placement area.

2.2.2 Pins

Modules are connected to each other through pins. For each pin $p \in \mathcal{P}$ we let $c(p) : \mathcal{P} \to \mathcal{MA}$ be the module p is connected to. Pins are placed with offsets on the modules so for $p \in \mathcal{P}$ we let $(\text{ofs}_x(p), \text{ofs}_y(p))^t \in \mathbb{Z} \times \mathbb{Z}$ be the offset of p with respect to the lower left corner of the module $c(p)$. Pins are only allowed within the boundary of their respective module.

2.2.3 Nets

Nets are subsets of $\mathcal{P}$ so for each $n \in \mathcal{N}$ we have $n \subseteq \mathcal{P}$. Further all pins should be part of a net so we have $\bigcup_{n \in \mathcal{N}} n = \mathcal{P}$. Without loss of generality we will also assume that only one net is connected to each pin; $n_1 \cup n_2 = \emptyset$ for $n_1, n_2 \in \mathcal{N}$. To simplify discussions we may say for a module $m \in \mathcal{MA}$ that $m \in n$ if and only if there exists a $p \in n$ such that $c(p) = m$. To each net we also assign a weight $w(n) > 0$ that may be used to describe the "importance" of that net.

This notation induces a hypergraph with modules as vertices and nets as edges which allows us to define:

Definition 2.1. *Connection graph* The connection graph of a circuit $\mathcal{C} = (\mathcal{M}, \mathcal{P}, \mathcal{N}, \mathcal{A})$ is the hypergraph $H = (N, M)$, where each $m \in \mathcal{M}$ is represented by a node $m' \in M$ and each net $n \in \mathcal{N}$ is represented by a hyperedge $h \in N$ and $m' \in h$ if and only if there exists $p \in n$ such that $c(p) = m$.

Note also that without offsets pins are redundant.

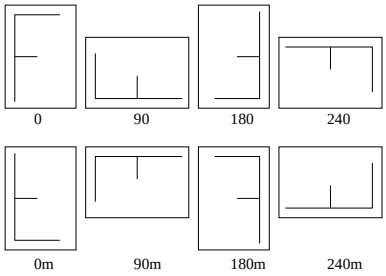


Figure 2.2: *The eight possible orientations of a module. In top row the module has been rotated 0, 90, 180 and 270 degrees respectively. In the second row the module is mirrored along the vertical axis.*

2.2.4 Placement

A placement of a circuit $\mathcal{C}$ describes the position and orientation of each module. Modules may assume one of eight orientations although in some cases modules are not allowed all orientations. Let

$$\mathbb{O} = \{0°, 90°, 180°, 270°, 0°m, 90°m, 180°m, 270°m\} \tag{2.1}$$

be the set of the possible orientations of a module (see figure 2.2).

We use the following convention. A placement P is given by $P = (x, y, o)$, where $x : \mathcal{MA} \to \mathbb{Z}$, $y : \mathcal{MA} \to \mathbb{Z}$ and $o : \mathcal{MA} \to \mathbb{O}$ are maps which describe lower-left coordinates and orientation of every module $m \in \mathcal{MA}$.

Definition 2.2. Overlap and containment We say that two modules $m_1, m_2 \in \mathcal{MA}$ overlap with respect to a placement $P = (x, y, o)$ if and only if

$$\begin{aligned} x(m_1) + w(m_1) > x(m_2) \quad \text{and} \quad x(m_2) + w(m_2) > x(m_1) \\ y(m_1) + h(m_1) > y(m_2) \quad \text{and} \quad y(m_2) + h(m_2) > y(m_1) \end{aligned} \tag{2.2}$$

Note that modules may abut.

Further we say that a module $m_1 \in \mathcal{MA}$ is contained within a module $m_2 \in \mathcal{MA}$ if and only if

$$\begin{aligned} x(m_1) \geq x(m_2) \quad \text{and} \quad x(m_1) + w(m_1) \leq x(m_2) + w(m_2) \\ y(m_1) \geq y(m_2) \quad \text{and} \quad y(m_1) + h(m_1) \leq y(m_2) + h(m_2) \end{aligned} \tag{2.3}$$

Definition 2.3. Legal placement A placement $P = (x, y, o)$ is *legal* if it obeys the following constraints:

- No two modules $m_1, m_2 \in \mathcal{M}$ overlap.
- All modules $m \in \mathcal{M}$ are contained within the placement area $\mathcal{A}$.
- For all fixed modules $m \in \mathcal{F}$: $(x(m), y(m))^t = (x_f(m), y_f(m))^t$.

We will refer to the first two constraints as the no-overlap constraints.

2.2.5 Comments on Orientations

Since a module may assume up to eight different orientations we should really take this into consideration when describing a legal placement. However the notation becomes too complex and in general for a placement $P = (x, y, o)$ we will assume:

- The width and height of a module $m \in \mathcal{MA}$ is with respect to the orientation $o(m)$ of m.
- The offset of a pin $p \in \mathcal{P}$ is with respect to the orientation $o(c(p))$ of $c(p)$.

We will not elaborate on determining width and height of a module or pin positions with respect to an orientation since this it is relatively easy to deduce.

2.2.6 More Formal Definition of the Placement Problem

We can now define the placement problem more formally.

Definition 2.4. *Placement problem* Given a circuit $\mathcal{C} = (\mathcal{M}, \mathcal{P}, \mathcal{N}, \mathcal{A})$ find a *legal* placement P that minimizes some cost function C which depends on $\mathcal{C}$ and P.

Cost functions The cost function may be one or a combination of several of the following:

- *Total wire-length (or netlength)* The length of wires necessary to connect pins in $\mathcal{C}$. There are a number of different ways to approximate the netlength and we will consider them in the following. If this is the cost function then we wish to minimize

$$\sum_{n \in \mathcal{N}} w(n) \cdot l(n), \tag{2.4}$$

  where $l(n)$ is the length of the wire connecting net n. The wire-length of nets is referred to as netlength.

- *Congestion* The number of wires in any one area. Congestion is loosely defined and some writers consider the number of wires in small areas of the placement area (see e.g. [71, 61]).

- *Area* The area may be minimized. There are two approaches to this. Either $w(\mathcal{A}) \cdot h(\mathcal{A})$ or $w(\mathcal{A}) + h(\mathcal{A})$ may be minimized. Note that with this objective function the size of $\mathcal{A}$ is not static.

- *Timing* Timing driven placement aims at minimizing the wire-length of critical paths. For timing driven optimization see e.g. [18, 42].

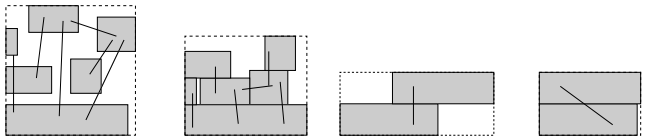


Figure 2.3: *(a) There is some correlation between minimizing wire and area and congestion. (a) The circuit on the left is minimized with respect to wire on the right. (b) However in some cases the objective functions are contradicting. On the left the circuit is minimized with respect to wire. On the right with respect to area.*

There is a consensus that the cost functions are in general correlated, e.g. minimizing wire will at least to some degree reduce area and congestion (see figure 2.3(a)) however in some cases the result may not be equal (see figure 2.3(b)).

The most widely used cost function is minimization of total wire-length and therefore it is also the cost function we have chosen.

We now prove the following:

Theorem 2.1. *The placement problem as described in definition 2.4 is NP-hard.*

Proof. To prove this we first introduce the following definition

Definition 2.5. 2-Partition decision problem (2PDP) Given a set S of integers decide if S can be partitioned in two sets A and B – $A \cap B = \emptyset$, $A \cup B = S$ – such that $\sum_{x \in A} x = \sum_{x \in B} x$.

It is well-known that 2PDP is NP-complete (see e.g. [13]).

We now define the decision problem for placement.

Definition 2.6. Placement decision problem (PDP) Given a placement problem as defined by definition 2.4, decide if there exists a feasible placement which satisfies the constraints of definition 2.3.

We first prove that PDP is NP-complete. A trivial algorithm with quadratic running time can determine if all modules are within the containment area, compare all pairs of modules for overlap and determine if all fixed modules are positioned correctly. Therefore PDP is in NP.

We now reduce from 2-partition. Assume we have an instance of 2-partition. It easy to see that for $\sum_{x \in A} x = \sum_{x \in B} x$ we must require that: $\sum_{x \in A} x = \sum_{x \in B} x = \frac{1}{2} \sum_{x \in S} x$. Now create a placement area with height 2 and width $\frac{1}{2} \sum_{x \in S} x$. For each element of $x \in S$ create a module which has height 1 and width x. This reduction is polynomial. Disallow rotation of modules. If we can find a feasible solution to this placement problem we must also have a feasible solution to the 2-partition problem. To see this observe that a solution to the placement problem must have two rows of modules,

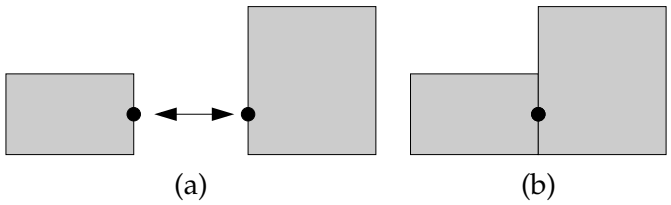


Figure 2.4: *(a) Two modules with pins opposite to each other (the black circles). (b) According to our formulation the pins may be placed coincident which would cause a short circuit.*

each corresponding to sets for which the sum of elements equals half the sum of all elements in S. Therefore PDP is NP-complete.

Since PDP is NP-complete and is the decision problem of the placement problem the placement problem must be NP-hard. □

2.2.7 Simplified Assumptions

The previous formulation is fairly simple and more constraints may be considered.

Spacing and rows In many cases spacing may be required between modules. E.g. standard cell instances often require a spacing between rows to accommodate routing if only one layer is used. Also the module positions may be confined to rows. We will consider spacing between modules.

Confinement to regions Some modules may be required to be placed in a certain *region* of the placement area or at the boundaries. This we will not consider but fixed modules will be considered.

Short circuits In the above definition of pins and modules, we may place two modules with pins at the same location (see figure 2.4) which would create a short circuit. This is unfortunate and the formulations can be altered to accommodate for this. However our formulation is simple and intuitive and the test circuits often requires spacing anyway. Also the amount of required spacing is not a trivial matter since it depends on the pin sizes. A *real* placement tool must take spacing more serious though.

It should be noted that IO-pins on the placement area are usually placed on a different layer than pins of modules.

2.3 Net Models

In this theses we have chosen to minimize the netlength. Since routing is itself an NP-hard problem a fast heuristic to estimate the netlength during the placement phase is necessary. This section is devoted to the most popular estimates. In the subsequent discussing we assume that a placement $P = (x, y, o)$ for a circuit $\mathcal{C} = (\mathcal{M}, \mathcal{P}, \mathcal{N}, \mathcal{A})$ is given and the netlength is to be measured.

2.3.1 Distance Metrics

Netlength depends on distance. For two points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^2$ we use the L_k-norm induced distance metric given by:

$$d_k(\mathbf{p}, \mathbf{q}) = \sqrt[k]{|p_x - q_x|^k + |p_y - q_y|^k}, \qquad \mathbf{p}, \mathbf{q} \in \mathbb{R}^2, k \in \mathbb{N}, \tag{2.5}$$

where $|x|$ is the absolute value of $x \in \mathbb{R}$, d_1 is the Manhattan or rectilinear distance between $\mathbf{p}$ and $\mathbf{q}$ and d_2 is the Euclidean distance between $\mathbf{p}$ and $\mathbf{q}$. Further letting $k \to \infty$ one gets:

$$\lim_{k \to \infty} d_k(\mathbf{p}, \mathbf{q}) = \max(|p_x - q_x|, |p_y - q_y|), \tag{2.6}$$

which allows us to define the L_∞ induced metric

$$d_\infty(\mathbf{p}, \mathbf{q}) = \max(|p_x - q_x|, |p_y - q_y|), \tag{2.7}$$

Absolute coordinates of pins In order to keep the notation simple we need some auxiliary notation. For $p \in \mathcal{P}$ let $\mathbf{A}(p) = (x(c(p)) + \mathrm{ofs}_x(p), y(c(p)) + \mathrm{ofs}_y)^t$ (i.e. the absolute coordinate of a pin p).

2.3.2 Rectilinear Steiner Tree and Minimum Spanning Tree

The routing-phase of the chip consists of connecting pins from the same net using minimum netlength. In modern design routing is done on separate layers. Obviously the total minimal netlength is achieved by connecting pins in each net with a steiner tree such that no two steiner trees intersect. Therefore a lower bound on the netlength is given by the sum of the sizes of the steiner minimal trees.

Currently only horizontal and vertical wires are used although more directions may be used in the future (see *www.xinitiative.org*). Therefore the smallest wire-length is achieved using a *rectilinear steiner minimal tree* (RSMT) (see figure 2.5(a)). An upper bound on the RSMT is the *rectilinear minimum spanning tree* (see figure 2.5(b)). Hwang

proved in [40] that the RMST of a set of n points in the plane is no longer than $\frac{3}{2}$ times the RSMT and the bound is tight for $n > 2$.

Calculating the RSMT is an NP-hard problem however most of the nets in the benchmark circuits contain less than 10 pins so exact algorithms [90] can solve the problem efficiently. Calculating the RMST can be done in $O(n \log n)$ time. Although not impossible to calculate the RSMT or the RMST bound most approaches to the placement problem estimates the wire-length more efficiently. Therefore we will present three other net models in the following. These are also the most popular net models and each of them has advantages as well as disadvantages. Net models are not the primary focus of this thesis and for more a more complete survey we recommend section 3.4 of [24].

2.3.3 Clique

The clique model arises from the graph representation of the VLSI-problem discussed in section 2.1.2. In the clique net model each net (or hyperedge) is converted to a clique subgraph (see figure 2.5(c)). The length of the net $n \in \mathcal{N}$ is then the *total* distance of pairs of pins in the net:

$$CL_k(n) = \frac{1}{2(|n|-1)} \sum_{p \in n} \sum_{q \in n} d_k(\mathbf{A}(p), \mathbf{A}(q))^k \tag{2.8}$$

The purpose of the $\frac{1}{|n|-1}$ multiplier is to prevent large nets from dominating the net model and the purpose of dividing with two is to count each "edge" once.

The clique netlength may be evaluated in time $O(|n|^2)$.

Relation to RSMT According to Vygen [87] the following holds:

Theorem 2.2. *For a net n containing $|n|$ pins the following is true.*

$$\begin{array}{lllll} \text{for } |n| \leq 3: & RSMT(n) & = CL_1(n) & & \\ \text{for } |n| = 4: & \frac{8}{9} RSMT(n) & \leq CL_1(n) & \leq \frac{|n|^2}{4(|n|-1)} RSMT(n) \\ \text{for } |n| \geq 5: & RSMT(n) & \leq CL_1(n) & \leq \frac{|n|^2}{4(|n|-1)} RSMT(n) \end{array} \tag{2.9}$$

Proof. See [87]. □

This shows that $CL_1(n)$ is certainly an upper bound of the RSMT.

2.3.4 Star

The star model is similar to the clique model but arises from converting hyperedges to star subgraphs (see figure 2.5(d)).

$$ST_k(n) = \min_{\mathbf{st_k}(n)\in\mathbb{R}^2} \sum_{p\in n} d_k(\mathbf{A}(p), \mathbf{st_k}(n))^k. \tag{2.10}$$

The point $\mathbf{st_k}(n)$ is called the *star-point* since this is exactly the position of the star-point of section 2.1.2.

$\mathbf{st_1}(n)$ is the median of respectively the x- and y-coordinates of the pins and can be determined in linear time (see chapter 10 of [13]).

Determining $\mathbf{st_2}(n) = (st_{2x}(n), st_{2y}(n))^t$ is also simple. We differentiate in x and y-coordinate to get the point which minimizes the sum. For the x-coordinate we get:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}st_{2x}(n)} \sum_{p\in n} d_2(\mathbf{A}(p), \mathbf{st}_k(n))^k \qquad (2.11)\\
&= \frac{\mathrm{d}}{\mathrm{d}st_{2x}(n)} \sum_{p\in n} \left(\sqrt{(x(c(p)) + \mathrm{ofs}_x(p) - st_{2x}(n))^2 + (y(c(p)) + \mathrm{ofs}_y(p) - st_{2y}(n))^2}\right)^2\\
&= \frac{\mathrm{d}}{\mathrm{d}st_{2x}(n)} \sum_{p\in n} (x(c(p)) + \mathrm{ofs}_x(p) - st_{2x}(n))^2\\
&= \sum_{p\in n} 2((x(c(p)) + \mathrm{ofs}_x(p) - st_{2x}(n))\\
&= \sum_{p\in n} 2(x(c(p)) + \mathrm{ofs}_x(p)) - 2|n|st_{2x}(n)
\end{aligned}$$

A similar result holds for the y-coordinate. This allows to determine $st_{2x}(n)$ and $st_{2y}(n)$ since minimum must occur for respectively $\frac{\mathrm{d}}{\mathrm{d}st_{2x}(n)} \sum_{p\in n} d_2(\mathbf{A}(p), \mathbf{st}_k(n))^k = 0$ and $\frac{\mathrm{d}}{\mathrm{d}st_{2y}(n)} \sum_{p\in n} d_2(\mathbf{A}(p), \mathbf{st}_k(n))^k = 0$:

$$\begin{aligned}
st_{2x}(n) &= \frac{1}{|n|} \sum_{p\in n} \mathbf{A}(p)_x\\
st_{2y}(n) &= \frac{1}{|n|} \sum_{p\in n} \mathbf{A}(p)_y
\end{aligned} \tag{2.12}$$

For $p = 1$ and $p = 2$ $ST(n)$ can be calculated in time $O(|n|)$ since determining $\mathbf{st_1}(n)$ and $\mathbf{st_2}(n)$ can be done in linear time and calculating $ST_k(n)$ can be done in linear time when $\mathbf{st_k}(n)$ is known.

Relation to RSMT $ST_k(n)$ is related to $CL_k(n)$ and this we use to prove the following:

Theorem 2.3. *For a net n containing $|n|$ pins the following is true.*

$$\begin{array}{lllll}
i) & \text{for } |n| \geq 1: & RSMT(n) & \leq ST_1(n)\\
ii) & \text{for } |n| \leq 3: & RSMT(n) & = ST_1(n)\\
iii) & \text{for } |n| \geq 4: & ST_1(n) & \leq \frac{|n|}{2} RSMT(n)
\end{array} \tag{2.13}$$

Proof. i) Since $ST_1(n)$ measures rectilinear distance between points and is therefore equivalent to a rectilinear steiner tree with one steiner point $RSMT(n) \leq ST_1(n)$ must hold.

ii) For $|n| = 1$ and $|n| = 2$ this is trivial. For $|n| = 3$ $st_1(n)$ must be a steiner point due to its definition.

iii) First we prove the following lemma.

Lemma 2.1. *For a net n we have:*

$$ST_k(n) \leq \left(2 - \frac{2}{|n|}\right) CL_k(n). \tag{2.14}$$

Proof. We have:

$$\begin{aligned} CL_k(n) &= \frac{1}{2|n|-2} \sum_{p \in n} \sum_{q \in n} d_k(\mathbf{A}(p), \mathbf{A}(q))^k \\ &\geq \frac{1}{2|n|-2} \sum_{p \in n} \sum_{q \in n} d_k(\mathbf{st_k}(n), \mathbf{A}(q))^k \\ &= \frac{|n|}{2|n|-2} ST(n) \\ &= \frac{1}{2 - \frac{2}{|n|}} ST(n) \end{aligned} \tag{2.15}$$

(The inequality in the second lines follows from the choice of $\mathbf{st}_k(n)$ which is chosen such that the inner sum is minimized). □

Now using theorem 2.2 and the previous lemma we get:

$$\begin{aligned} ST_1(n) &\leq \left(2 - \frac{2}{|n|}\right) CL_1(n) \\ &\leq \left(2 - \frac{2}{|n|}\right) \frac{|n|^2}{4|n|-4} RSMT(n) \\ &= \left(\frac{|n|^2 - |n|}{2|n|-2}\right) RSMT(n) \\ &= \frac{|n|}{2} RSMT(n) \end{aligned} \tag{2.16}$$

□

2.3.5 Bounding-Box

The bounding-box BB netlength is by far the simplest. The netlength estimate for a net n is simply the half-perimeter of the bounding-box surrounding the net (see figure

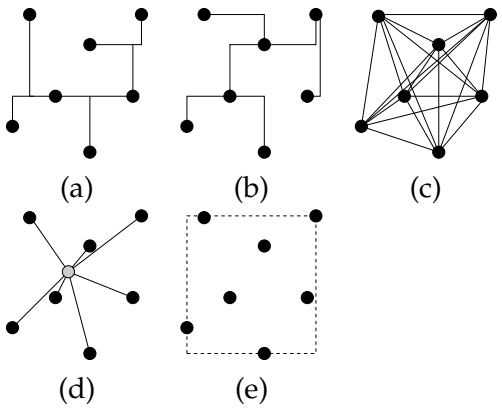


Figure 2.5: *The five net models. (a) Rectilinear Steiner Minimal Tree. (b) Rectilinear Minimum Spanning Tree. (c) Quadratic Clique (CL_2). (d) Quadratic Star (ST_2). (e) Bounding-box BB. The black circles are pins. The extra circle in (d) is the star point.*

2.5(e)):

$$BB(n) = \max_{p \in n}(A_x(p)) - \min_{p \in n}(A_x(p)) + \max_{p \in n}(A_y(p)) - \min_{p \in n}(A_y(p)) \tag{2.17}$$

Calculating $BB(n)$ can easily be done in time $O(|n|)$ although updating the netlength with respect to movement of modules may be done faster.

Relation to RSMT Chung and Graham published a proof in [12] which stated that $RMST(n) \leq \frac{\sqrt{|n|}-1}{2} BB(n)$. However Brenner and Vygen discovered an error in the proof in [7]. In [24] Farø proved the less tight ratio:

$$RMST(n) \leq \frac{\lceil \sqrt{|n|} \rceil + \frac{3}{2}}{2} BB(n) \tag{2.18}$$

However for $|n| \leq 3$ it is easy to see that $RSMT(n) = BB(n)$.

Variations on bounding-box netlength In [9] Caldwell et al. presents extensions to the bounding-box netlength. Their primary focus is on estimates of the netlength when the positions of the modules are only determined within regions. However they also demonstrate how the bounding-box *area* combined with height/width *ratio* can be used as an estimator for the RSMT.

2.4 Minimizing Quadratic Netlengths

In appendix C.1 we show how both the quadratic clique and star netlengths can be expressed by matrix notation. If the corresponding problem is relaxed by removing

the no-overlap constraints we can minimize the quadratic netlength by solving two independent problems of the form:

$$\min_{\tilde{\mathbf{x}} \in \mathbb{R}^{|\mathcal{M} \setminus \mathcal{F}|}} g(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} + \mathbf{d}^t \tilde{\mathbf{x}} + f, \tag{2.19}$$

with $\mathbf{C} \in \mathbb{R}^{n \times n}$, $\mathbf{d} \in \mathbb{R}^n$ and $f \in \mathbb{R}$ defined according to theorem C.1 or C.2 ($n \in \mathbb{N}$ varies).

According to multi-variable calculus a local minimum or any extreme point of g must occur where $\nabla g(\tilde{\mathbf{x}}) = \mathbf{0}$ for $\nabla = (\frac{\partial}{\partial \tilde{x}_1}, \ldots, \frac{\partial}{\tilde{x}_{|\mathcal{M} \setminus \mathcal{F}|}})$. Using the fact that $\mathbf{C}$ is symmetric (theorem C.3) it is easy to see that

$$\nabla g(\tilde{\mathbf{x}}) = \mathbf{C}^t \tilde{\mathbf{x}} + \mathbf{C} \tilde{\mathbf{x}} + \mathbf{d} = 2\mathbf{C} \tilde{\mathbf{x}} + \mathbf{d} \tag{2.20}$$

Further one can show that this extreme point is a minimum (see [77]). So minimizing g is equal to solving the equation system:

$$2\mathbf{C} \tilde{\mathbf{x}} = -\mathbf{d} \tag{2.21}$$

By using Gaussian elimination we may invert $\mathbf{C}$ in time $O(n^3)$. This is unsatisfactory and it turns out there is a faster heuristic way.

2.4.1 The Conjugate Gradient Method

The Conjugate Gradient Method ([35, 74]) works on parabolic functions like g by taking steps towards the global minimum. The method requires that $\mathbf{C}$ is symmetric and positive definite which is proven in appendix C.1. It is an iterative procedure and in each step the current solution is moved in the direction of the eigenvalues of $\mathbf{C}$. When the solution is deemed close to minimum the procedure ends. We will not describe the method in detail here. Instead we will refer to the excellent introduction by Schewchuk [77]. The order of convergence of the method depends on the spectral condition number k of $\mathbf{C}$ and is $O(\sqrt{k}m)$ for $\mathbf{C} \in \mathbb{R}^n$ when m is the number of non-zero elements of $\mathbf{C}$ and sparse matrix data structures are used.

The matrix $\mathbf{C}$ is in general sparse. Each row corresponds to a module. For the star model the number of non-zero elements in a row is roughly equal to the number of *nets* the module of that row is connected to. For the clique model this number is equal to the number of *modules* the module is connected to. It is generally assumed and in section 5 we will show empirically that the average number of nets each module is connected to is usually less than 5. Therefore if one uses sparse matrix data structures the size of the matrix $\mathbf{C}$ is the number of non-zero elements which in practice is $m = O(|\mathcal{M} \setminus \mathcal{F}|)$.

Preconditioning As mentioned above the Conjugate Gradient Method depends on the spectral condition number of $\mathbf{C}$. This can be improved by using a preconditioner. The essence of the preconditioning method is that instead of solving a system of the form $\mathbf{Ax} = b$ one can solve an auxiliary system $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}b$. If $\mathbf{M}^{-1}\mathbf{A}$ is better conditioned than $\mathbf{A}$ we may get faster convergence. The matrix $\mathbf{M}$ is called a preconditioner and the problem is of course determining $\mathbf{M}$ such that we can easily find $\mathbf{M}^{-1}$. Ideally one would pick $\mathbf{A}$ but since we do not know $\mathbf{A}^{-1}$ we have to make another choice. The simplest preconditioner is the diagonal of $\mathbf{A}$ and the most popular is an incomplete cholesky factorization of $\mathbf{A}$. We will not discuss preconditioning further here but even diagonal preconditioning can improve the order of convergence significantly.

Reducing the number of non-zero elements A simple way to reduce the number of non-zero elements of the matrix $\mathbf{C}$ is to consider the cardinality of nets. For the clique formulation nets with n elements contribute with $n^2 - n$ non-zero elements to $\mathbf{C}$ if we disregard the diagonal. Similarly nets with n elements in the star formulation contribute with $2n$ non-zero elements disregarding the diagonal. Since there are many nets with cardinality two (see section 5) we can reduce the number of non-zeroes of $\mathbf{C}$ by using the clique formulation for nets with cardinality two and star formulation for nets with higher cardinality.

2.5 Minimizing Linear Netlength

The unconstrained linear bounding-box formulation can also be solved in efficient time. Here the objective function is the dual of a minimum-cost-flow problem (see C.2 for details). The dual problem can be solved and by using LP-duality a solution to the primal problem can be determined. Weis and Mlynski considered this in [91]. Unfortunately the fastest known maximum-flow algorithm takes $O(mn\log(n^2/m))$ time[3] (see [27]) and therefore has quadratic running time.

[3]For the VLSI-design problem m is the number of modules+nets and n is the number of pins+nets

3 Previous work

The VLSI-placement problem has been the focus of much research over the past 20-30 years. The number of papers on the subject is overwhelming, and a complete survey is certainly out of the scope of this thesis. Therefore we have chosen to concentrate on the more interesting and relevant research of the last decade.

In the literature the placement problem is divided roughly into these five categories:

- **Global placement** deals with assigning modules to positions on the placement area close to their optimal position. Some writers use the term "relative placement". In general global placement generates an initial placement with overlap. A global placement heuristic can be both constructive and iterative.
- **Final placement** optimizes the positions of the modules with a global placement as initial solution. In general this is an iterative procedure which moves between solutions. In some cases only legal solutions are considered however often overlapping solutions are accepted during the improvement step. The result of the final placement stage is usually a non-overlapping *final* placement.
- **Area minimization** The area minimization problem is a two-dimensional packing problem with rectangular shapes. The area minimization problem is NP-hard.
- **Legalization** If the final placement is illegal some form of legalization must be conducted at the end.
- **Post optimization** Generally a legal solution may be improved slightly without changing relative order of the modules.

The literature on the five topics is concentrated mainly on global placement, final placement and area minimization. Some solution methods are presented with both a global placement and corresponding final placement heuristic.

These categories will form the outline of this chapter and a section is devoted to each of the topics.

On Standard-cells and General-Cells In general most work has been done with respect to standard-cells. The reason for this seems to be that general-cells renders the placement problem too difficult for current placement heuristics. Although this thesis aims at new algorithms for *general-cells* there is at least one reason for presenting the standard-cell heuristics. Well-functioning heuristics for standard-cells may work well with general-cells and in fact some of the heuristics allow large macro-blocks as part of the problem instances. The standard-cell heuristics could therefore serve as inspiration.

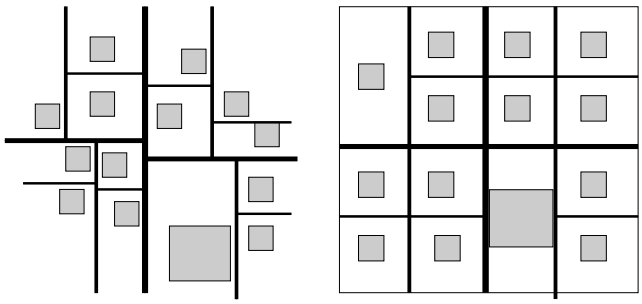


Figure 3.1: *The graph partitioning method. (a) The circuit is partitioned. (b) With each partition the placement area is divided into two equal sized parts and the modules of of each part is confined to the corresponding half. The thickest lines are the earliest bisections.*

3.1 Global Placement

There are surprisingly few overall approaches to global placement. Three main ideas dominate. Partitioning based on the connection graph, partitioning based on analytic placement and complete analytic methods.

3.1.1 Graph Partitioning

A classic global placement method is based on hypergraph partitioning. The hypergraph partitioning in its most basic form bisects the connection graph (see section 2.2.3) recursively. Nodes in the connection graph are usually given weight corresponding to the area of their module. Each bisection is chosen such that the weight of nodes (modules) on either side is close to equal while at the same time the number of hyperedges cut by the bisection is kept at a minimum. This optimization problem is indeed NP-hard (reduce from 2-partition) because of the balance requirement.

The thought behind graph-partitioning is that highly-connected components of the hypergraph should be close to each other in the final layout. When the recursion is complete, e.g. only one node left, a relative placement of the modules is extractable by the bisections (see figure 3.1). Note that another purpose of the graph-partitioning method is to reduce congestion; the number of nets crossing any "line" on the placement area is at least to some degree minimized.

The two most famous graph-partitioning heuristics are of Kernighan and Lin [47] and Fiduccia and Mattheyses [23]. Fiduccia-Mattheyses' is based on that of Kernighan and Lin and is the younger and more popular of the two. Fiduccia-Mattheyses' has also been the primary focus of later research (see e.g. [43, 75, 51, 52, 33, 14, 10]). Both heuristics are iterative and move-based. The Kernighan-Lin heuristic swaps modules on opposite side of the cut whereas Fidducia-Mattheyses' moves one module to the

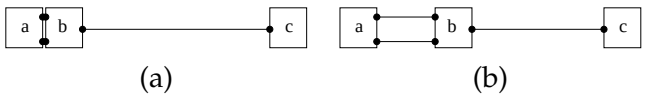


Figure 3.2: *Comparison of minimal linear and quadratic placements. (a) Linear formulation; module b is completely adjacent to a. (b) Quadratic placement. In Quadratic formulation the two nets on the left of b weigh twice as much as the net on the right.*

opposite side (the details are complicated). An obvious problem with the move-based heuristics is generating an initial solution.

The graph-partitioning method is simple but does not directly consider module coordinates or netlength. Further the min-cut objective seems somewhat inadequate at modeling congestion. Therefore we choose to abandon graph partitioning at this point. The complete literature and methods involved in graph- and circuit-partitioning is also far too vast to consider here. Instead we point to [2] which is a fine survey on graph-partitioning for VLSI-optimization.

3.1.2 Analytic and Relaxation Based Placement

In analytic placement the no-overlap constraints are disregarded or incorporated into the objective function. The problem is then a matter of minimizing an unconstrained objective function.

In section 2.4 and 2.5 it was explained how the unconstrained placement problem with quadratic and linear bounding-box netlength can be solved. The quadratic netlength formulations can be solved fast using the Conjugate Gradient Method or any similar numerical linear equations solver while the bounding-box formulation requires quadratic running time network-flow methods.

Comparison of linear and quadratic netlength for analytic placement Sigl et al. [79] and Mahmoud et al. [58] compare the linear formulation with the quadratic. The linear formulation will tend to stack modules on top of each other while the quadratic formulation will spread them slightly. Secondly the average standard deviation of the netlengths is smaller for the quadratic formulation; short nets in the linear objective are longer in the quadratic and long nets shorter. A simple example is shown on figure 3.2.

Figure 3.3 shows several examples of how much overlap minimizing the unconstrained bounding-box formulation results in compared to the star netlength. We have placed a small macro-cell circuit, a standard-cell circuit and a real-life circuit analytically ignoring no-overlap constraints using both bounding-box netlength and star netlength. The result is excessive stacking and unacceptable running times for the bounding-box

netlength. The bounding-box netlengths were minimized using CPLEX optimization software [4] although not using the maximum-flow formulation but the simple linear program formulation.

As it should be clear from figure 3.3 the decision between a linear and a quadratic objective function is not just a matter of choosing one or the other. In most global placement heuristics the quadratic function is chosen because of its speed and because the relative order of modules can used to generate a solution. This is impossible with the linear formulations because modules stack on top of each other. During local search however the linear bounding-box formulation is used because it is closer related to the RSMT.

Since the quadratic netlengths are easier to optimize numerical methods have been proposed that smooth the quadratic objective function and allows for extraction of relative position of modules while maintaining an almost linear objective function (see e.g. [1, 46, 4]). An example of a simple linearization scheme is given in section 3.1.3 during the description of Gordian. In general the linearization schemes are a couple of times slower than the equivalent quadratic solution methods, but can make a substantial difference in solution quality.

An interesting quality of the quadratic netlengths is that solutions to the unconstrained problem always lie within the left-most, right-most, top-most and lower-most pins. The explanation is simple. If a module was placed e.g. further left than the left-most pin, netlength could be reduced by moving it until it abutted with the pin. This observation is basis of several placement algorithms.

Modules as points The first analytic placement heuristics used the *modules as points* convention; i.e. all pins on a module are assumed at the center of the module. However we have found no need for this simplification; the analytic placement formulations are only slightly more complicated if pin offsets are considered. On the other hand considering the size of the placement area compared to the size of the modules this probably has very little effect on the solution quality. However it does simplify the problem for standard-cells since orientation of the modules is no longer an issue; all orientations are equally good.

What is apparent from figure 3.3 is that stand-a-lone analytic placement results in far too much overlap. Therefore placement-heuristics combine the analytic placement with module spreading techniques. The two most popular forms are partition based and force-based.

[4] CPLEX is created by ILOG Inc. (www.ilog.com)

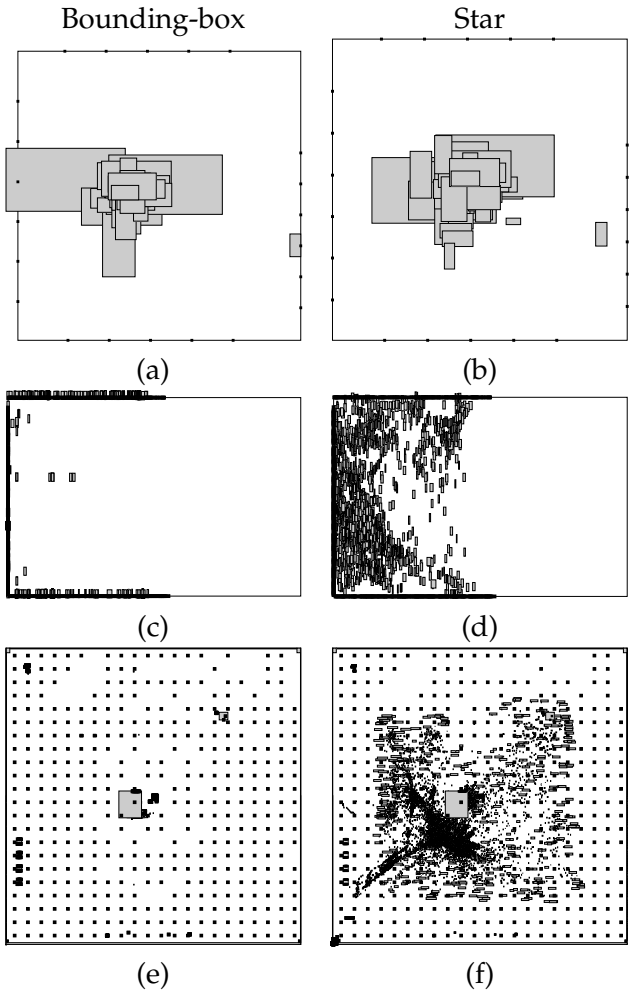


Figure 3.3: *Comparison of unconstrained placement of star wire-length and bounding-box wire-length on three benchmark circuits, ami49 (a and b), industry1 (c and d), clk (e and f). (a), (c) and (e) are generated based on the bounding-box linear formulation using CPLEX. (a) and (c) where generated in a few seconds, while (e) took more than* 10 hours. *(b), (d) and (f) were all generated using a Conjugate Gradient Method in less than 20 seconds. Note that the bounding-box placements of (a), (c) and (d) generate far more overlap than the corresponding star placements of (b), (d) and (f). This is especially easy to see on (c) and (e) where the modules pile together on very few spots or along the edges of the circuit. The completely black rectangles on the edges of ami49 and industry1 and uniformly distributed on clk are I/O-pins.*

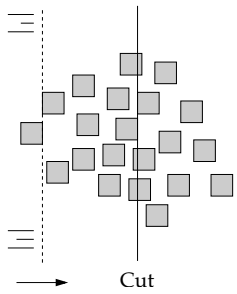


Figure 3.4: *The proud cutting method. An analytic placement is cut in two. The algorithm moves a cut-line from left to right (sweep-line) until the area on either side is roughly half.*

3.1.3 Analytic Based Partitioning Heuristics

The analytic partitioning heuristics are closely related to the graph-partitioning heuristics. But instead of minimizing cuts they use the analytic placement to guide the position of the cut-lines.

The oldest analytic method we have found in the literature is from 1970 and by Hall [34]. He considered quadratic placement with "slot" constraints. The slot constraint was added to the objective function as Lagrange-relaxation and in order to determine the Lagrange multiplier he did eigenvector analysis.

Proud

The eigenvector analysis approach was dropped by Tsay et al. [83] because it was far too slow on large circuits. Instead they use a quadratic clique formulation. Their idea was based on reformulating the problem as a linear resistive network problem which they optimize by solving a system of linear equations. This is done with Successive Over Relaxation (SOR) (see e.g. [8]). The method is a predecessor of optimizing by Conjugate Gradients.

As mentioned earlier such a placement is likely overlapping and therefore a partitioning method is used. A vertical cut line is moved from left to right until the sum of the area of the modules on the left of the cut line is roughly half (see figure 3.4). This divides the set of modules in two groups. The modules are now confined to two regions each corresponding to half the placement area and the method now optimizes each group while pretending that the modules of the opposite group are fixed; their coordinates are simply projected onto the region boundary. Proud is related to the graph partitioners but use the analytic placement to guide the cut instead of the min-cut objective.

The method proceeds recursively on each region alternating between vertical and horizontal cut lines until only one module remains. A heuristic four-way partitioner

is also proposed and test runs show that the four-way partitioner gives improved netlength but often uses twice the time or more.

Gordian

Gordian was presented in 1991 in [49] by Kleinhaus et al. and uses a variety of methods.

A clique net-model is minimized analytically to give a global placement. Gordian uses iterative improvement consisting of bi-partitioning and quadratic optimization.

Gordian works with regions. Initially the entire circuit is a region. In each iteration each region is split in two. The split is done with a cut-line method similar to the one used in Proud. However here both a horizontal and a vertical cut-line is found and the one with fewest crossing nets (smallest *cut*) is chosen unless some width/height ratio of the resulting regions is far from 1. To improve the cut the Fiduccia-Mattheyses min-cut heuristic is used to exchange modules on different sides of the cut-line.

In order to prevent the modules from moving outside their assigned region in an iteration Kleinhaus et al. uses a center of gravity constraint which requests that the center of gravity of the modules in each region is the center of the region. This constraint and analytical minimization of the netlength corresponds to a quadratic program with constraints. However by careful inspection of the constraints the objective function and constraints are combined to form a new unconstrained quadratic problem which can be solved by e.g. the Conjugate Gradient Method.

It may happen that modules in two subregions will migrate across a cut-line if the partition is bad – i. e. the center of gravity constraint is insufficient to keep modules in their respective regions. Therefore a repartition step is added. The subsets of modules from two neighboring regions which violate the region constraint are merged and repartitioned. This is repeated until all modules are within their region. Kleinhaus et al. concludes that in practice one repartition step suffices.

The iterative procedure is stopped when there are less than k cells in each region.

A legal placement of standard-cell circuits is achieved by sorting the modules according to their y-coordinate and splitting them into r rows. The sequence of the modules in the rows is determined by their x-coordinate.

Mixed-cell layout is also considered. Since there may be up to k cells in each region, macros are given special attention. The relative positions of the macros in the last quadratic solution is used with a heuristic which determines the minimal area slicing structure, thereby reducing the area required by the macros.

Gordian-L A linearization scheme was applied to Gordian by Sigl et al. in [79]. The quadratic clique netlength was changed to a pseudo-linear star netlength. Sigl et al.

simply used the fact that the linear star length is separable in x- and y-coordinates and that e.g. the x-coordinate contribution can be written as:

$$\mathbf{ST_1}(n)_x = \sum_{p \in n^p} \frac{(\mathbf{A}(p)_x - st_{1x})^2}{|\mathbf{A}(p)_x - st_{1x}|} \tag{3.1}$$

Sigl et al. approximated st_{1x} with st_{2x}. The analytic minimization algorithm now proceeds iteratively in the following manner. The sum of the denominators is calculated and inserted into the ordinary quadratic formulation as a constant. The quadratic formulation is solved and the new sum of denominators is calculated. This is repeated until the difference between linear netlengths from succeeding iterations is below some ϵ. Note that this analytic minimization requires repeated minimization of the quadratic function. The implementation is called Gordian-L. The algorithm improves the netlength of the two and three pin nets compared to Gordian. No run times are reported however.

The algorithm of Vygen

Another partitioning method is by Vygen [86]. It is primarily for standard-cells but it can also handle mixed-cell-layout.

First a quadratic placement is calculated by minimizing the clique netlength. Then the circuit is divided into four parts and the position of the modules from the quadratic solution is used to determine into which of the four parts each module is to be placed. The splitting algorithm gives close to minimal total movement of the modules and runs in linear time. The process is repeated but for the subsequent quadratic optimizations the nets are split so that no net cross a region. If a net crosses a boundary of a region an artificial module is placed on that boundary. This leads to a new formulation for each net on each region containing only artificial and ordinary modules of that region. (see figure 3.5).

Before partitioning at each level a re-partitioning step is also conducted on *each* 2×2 sub-grid in the current grid. The modules of the 2×2 sub-grid are "thrown" together again and a quadratic problem for the sub-grid is solved. After partitioning this leads to a new placement of the modules in the 2×2 sub-grid which is accepted if it is better than the old one.

Extra care is taken to avoid situations where all modules in the same region end in the same spot. This is done by introducing a constraint that makes the center of gravity of the modules be at the center of each region.

Vygen's algorithm can also place large macro modules in combination with standard-cells. This is done by including them in the partioning step. When regions become too small to contain the macros a branch-and-bound algorithm is used to place all macros within the region while minimizing total movement. The algorithm is similar to that of Onodera et al. [68] (see section 3.5), but here branching occurs on the

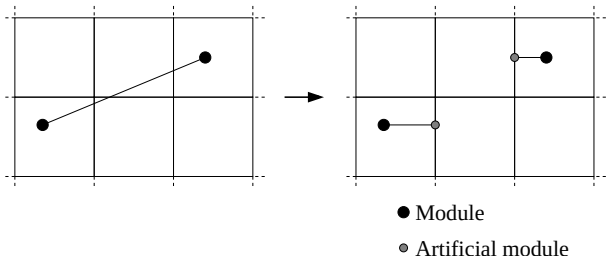


Figure 3.5: *Illustration of Vygen's splitting technique. Since the two modules are placed in separate regions the net between them is split in two. Artificial modules are placed at the region boundaries and connected to the module of their respective regions. The new formulation is used as basis of a quadratic optimization in the next iteration.*

two most overlapping modules. The nodes of the branch-and-bound algorithm are linear programs which are duals of minimum cost flow problems and can be solved in $O(n \log n(m + n \log n))$[5] time [69].

The final phase of the algorithm considers moving modules between the final regions. Regions are characterized as having too many modules if they cannot contain the modules. Also some regions may have surplus space. The movement of modules is modeled as a minimum-cost-flow problem between adjacent regions. A knapsack problem determines which modules to move.

3.1.4 Force-Based Methods

Some writes refer to all analytic placement methods as force-based methods. The name originates from an alternative interpretation of the analytic quadratic placement. If modules are modeled as objects and nets as springs connecting the objects, then by Hooke's law minimizing the netlength is equivalent to putting the "spring-system" into equilibrium (see figure 3.6).

In this text the we define the force-based methods as the successors of the method introduced in [19] by Eisenmann and Johannes in 1998. The force-based method uses the analytic placement techniques to achieve an illegal overlapping placement which minimizes quadratic netlength.

Eisenmann and Johannes' heuristic proceeds iteratively introducing "repelling forces" between each module and bins on the placement area with overlap (see figure 3.7). In each iteration the current overlapping forces are added to the quadratic netlength and the combined quadratic function is minimized.

The force at a location $(x, y)^t$ is set to:

$$\mathbf{f}(x, y) = \frac{k}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(x', y') \frac{\mathbf{r} - \mathbf{r'}}{|\mathbf{r} - \mathbf{r'}|^2} dx' dy', \tag{3.2}$$

[5] n is number of modules and m is number of "disjointness" constraints.

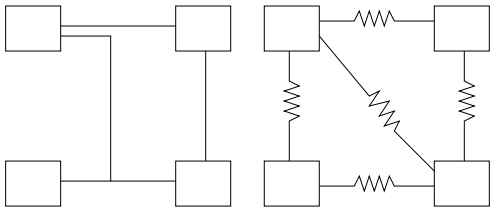


Figure 3.6: *Origin of "the force method". By reformulating the netlength (a) to a system of springs (b) minimizing the quadratic netlength is equivalent to bringing the spring-system into equilibrium.*

where $D(x', y')$ is the "density" of the point $(x', y')^t$ and $\mathbf{r} = (x, y)^t$ and $\mathbf{r}' = (x', y')^t$. The density function is a value for each bin which describes the number of modules in it. By careful inspection one notices that $\mathbf{f}(x, y)$ can be calculated at the center of every bin by a two-dimensional folding of two functions; $D(x', y')$ and $\mathbf{g}(x, y) = \frac{\mathbf{r}-\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|^2}$. By using a Fast Fourier Transform (see e.g. [67, 60]) we can evaluate $\mathbf{f}(x, y)$ at every bin center in $O(n \log n)$ time, where n is the number of bins. This detail was *omitted* from the original paper and seems to have been overlooked by at least the writers of [61], however a *personal communication* with Hans Eisenmann verified that this was indeed the method originally used.

To improve the netlength Eisenmann and Johannes also used a variation of the linearization method proposed in [1]. Finally the Domino local search method which will be discussed in section 3.2.2 was used for final placement.

Eisenmann and Johannes only used the heuristic on standard-cell circuits but Mo et al. [61] extended the force-based heuristic to macro-cells. Unlike Eisenmann and Johannes no linearization scheme is considered. Also instead of introducing repulsive forces a filling force is introduced between modules and empty regions of the placement area. Mo et al. also consider orientation of the macro-cells. For each cell the new orientation is determined by considering all eight possible orientations and selecting the one which minimizes the external force. Since changing orientation of one cell will affect the force on other cells, care must be taken not to change orientation of too many cells at once. Therefore only 10% of the cells are selected for orientation optimization in each iteration. Finally Routing and pad positioning are considered. Routing is considered by estimating congestion in each bin and pad positions are determined similar to module-position by including the pads in the quadratic netlength formulation, but limiting their movement to one dimension. The iterative flow is divided into three stages. Stage one gives initial positions of the macro-cells. Stage two optimizes orientation and routing. Finally stage three removes any additional overlap.

Hu and Marek-Sadowska [37] introduced a slight modification of Eisenmann and Johannes' method referred to as Fixed-point Addition and Relaxation (FAR). Instead of adding repelling forces from overlapping regions to the objective function they intro-

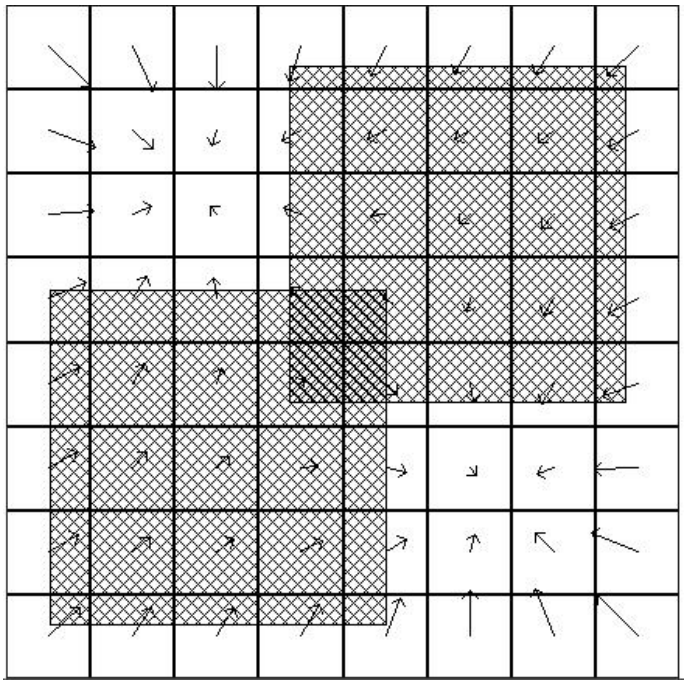


Figure 3.7: *The force method of Eisenmann and Johannes. The placement area is split into regions. At each region the force of the modules based on overlap and empty space in all other regions is calculated. Here the force vectors are shown as arrows. Notice that the force in the overlapping regions is towards regions without any cells. Also notice that the force formulation pulls cells toward the center of circuit. The image was created by our placement program as described in section 9.1.*

duced a pseudo-module (fixed-points) for each module and a pseudo-connection between each module and its associated pseudo-module. The advantage of fixed-points is that they are easier to control than the forces. Hu and Marek-Sadowska are able to confine the analytic placements to a specific region which is not as simple with the constant forces of Eisenmann and Johannes.

A completely different force-based method called Attractor Repellor Approach (ARP) was proposed in 1999 by Etawil et al. [20]. Instead of looking at overlapping regions a repelling force is introduced between directly connected modules. However this plain modification of the objective function is insufficient and results fall into two categories; too much cell overlap or poor netlengths because the objective function has been distorted too much. Therefore Etawil introduces attractive forces in low density regions.

3.1.5 Simulated Annealing for Global Placement

Simulated annealing was introduced by Kirkpatrick et al. [48] and, because of its simplicity, it has become the meta-heuristic of choice if all else fails. Naturally simulated annealing has also been applied to the placement problem, however most writers use the methodology for final placement because the solution space is considered too large during global placement.

Sarrafzadeh and Wang use simulated annealing on a global placement grid of bins. The method was first introduced in [76] (NRG). The placement area is partitioned into bins. Now each move in the simulated annealer is a swap of cells between bins. The size of the grid is determined from analysis of the circuit.

The method was also used in [89] (Dragon) – also by Sarrafzadeh and Wang – at the detailed stage of the global placement. The first stage of the placer in [89] consists of a hierarchical approach. The placement area is recursively divided into a 2x2 bin structure. This recursion ends when there are only 7 modules in each bin. At each level of the hierarchical algorithm a bin swapping algorithm attempts to swap the contents between neighboring bins to improve the placement. The details however are omitted from the article.

3.1.6 Clustering

Some placement algorithms combine modules in clusters (clustering). There are at least two reasons to consider this method.

- Clustering sub-circuits can reduce the running time since there are fewer modules.

- Clustering can move a simple placement algorithm out of local minimum because it considers more than simple individual cell movement (See figure 3.8)

The primary problem of clustering is determining clusters. There are two approaches to this problem:

- Determining the clusters strictly from the hypergraph representation of the circuits.
- Determining the clusters based on placement.

Clustering methods have been used with the graph partitioning algorithms of section 3.1.1 to improve the speed of the partitioning (see [32, 94]). However clusters have also been used in direct placement.

Grover and Mallela [30] use simulated annealing for global placement. In order to reduce the search space of global placement Grover and Mallela use a clustering algorithm which combines modules with high connectivity together and then use simulated annealing to optimize the clusters' positions on the placement area. The clustering algorithm starts with a seed module. Now a set of candidate modules is searched and candidates are rated according to several criteria (e.g. number of additional pins, common nets with the seed, etc.). Grover and Mallela even allow clusters to be combined.

The TimberWolf Placer version 7 by Sun and Sechen [80] also uses simulated annealing in conjunction with clustering but with a different clustering algorithm. Here a predefined number of clusters and a lower and upper capacity of the clusters is specified. The algorithm assigns weights to clusters depending on the fraction of pins from nets belonging to the cluster. The total weight is optimized using a separate simulated annealer. The method shows quite promising results especially for large circuits. Although no run time comparison between the clustering method and an ordinary simulated annealing is reported, the authors state that the clustering method can be up to 7 times faster.

Both of the previous clustering based heuristics use the hypergraph to determine clusters however Hur and Lillis [38] use the current placement. Clusters are optimized many at a time. Some of the clusters are extracted and the remaining modules are fixed at their current position. Hur and Lillis now use a concept they refer to as Relaxation Based Local Search (RBLS). The no-overlap constraints are relaxed and the resulting linear program is solved using network-flow methods. Whenever Hur and Lillis reach local optimum with a final placement algorithm they apply their clustering method which is quite effective at escaping local optimum.

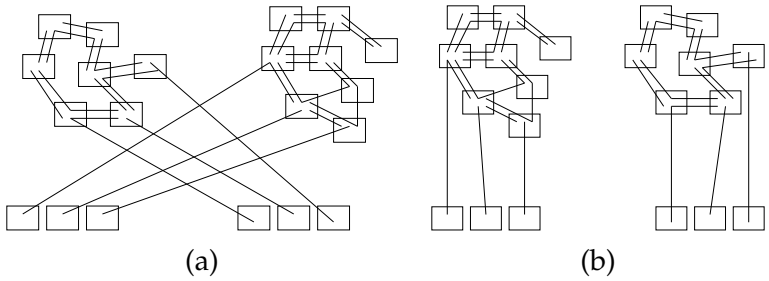


Figure 3.8: *Illustration of how clustering may improve an otherwise local minimum. (a) A swap-based local search cannot improve the placement because the modules of the two clusters are too strongly interconnected. However swapping the entire clusters (b) improves the netlength. This is of course a very simple example.*

3.1.7 Other Strategies

Although the previous sections cover most of the successful approaches from the literature other interesting approaches have been attempted.

Relaxation Based Local Search

In [39] (Mongrel) Hur and Lillis use the the Relaxation based local search method introduced in [38]. Here they use a two-dimensional bin structure the placement area with bin capacities. They now consider a *global placement* legal if it satisfies the bin capacities. Also module coordinates during the global placement are simply the center of the bin they belong to.

It is not clear from [39] how they form an initial solution but the global optimization is the following iterative process: In each iteration a sub-circuit of m modules is extracted. The remaining cells are fixed at their current positions. Now the linear programming formulation of the sub-circuit is optimized using network flow methods which Hur and Lillis presented in [38], and a relaxed placement is constructed.

In order to remove overlap of the new placement a legalization step is introduced. Hur and Lillis moves each module from the sub-circuit to its new relaxed position. If that position is legal it is accepted. If it is illegal (bin capacities exceeded) they use a legalization method they call "node-rippling". The "node-rippling" moves cells from a bin which exceeds capacity constraints to neighboring bins. Careful analysis determines the best candidates for movement to keep netlength increase low. The result of the "node-rippling" method is a sequence of moves of modules which will legalize the placement.

A very important element of Hur and Lillis heuristic is that they consider m legal placements of each iteration and choose the best. This increases the number of considered solutions.

To further improve on the placement Hur and Lillis also use the min-cut heuristic of Fidducia and Mattheyses (see section 3.1.1) between adjacent bins.

3.2 Final placement

Final placement has been given surprisingly little attention. Many global placers are combined with a simple final placement method which swaps module positions, often in a simulated annealing framework. Most final placement heuristics are limited to standard-cells.

3.2.1 Simulated Annealing

The final placement heuristic adopted by most authors is based on simulated annealing. There are two different approaches:

- Optimize netlength and disregard the no-overlap constraint between modules but penalize overlap in the objective function.
- Optimize netlength but when moving modules to new locations ensure that the resulting placement is still legal.

The first method has been adopted by [78] and [76] (NRG). In both cases the simulated annealing considers swap of modules and moving modules to new positions.

The second method has been adopted by [80] and [29]. Note that [78] constitute TimberWolf 6.0 and [80] constitute TimberWolf 7.0 and are written by the same authors. The conclusion the authors give in [80] is that it is better to update the netlength to fit the new legal placement. Since this update is very expensive – up to half the modules of a row can be moved left or right – they develop an approximate method which is an improvement of a method developed in [29]. The main idea is to look at small moves of each module. For each module m a value, *prefer*, is initialized to 0. *prefer* is decreased for each of the nets for which m is rightmost and increased for each of the nets for which m is leftmost. By multiplying with the distance m is moved an approximate new netlength is calculated (see figure 3.9).

3.2.2 Greedy Approaches

The Domino-improver by Doll et al. [15] works with legal placements but can accept an overlapping placement as initial solution. The current placement is split into regions. Each region is placed one at a time. The region p is placed as follows: Other regions are considered fixed. Domino works from left to right, so all modules to the left of p have been placed in this iteration. Each cell is now split into sub-cells of equal

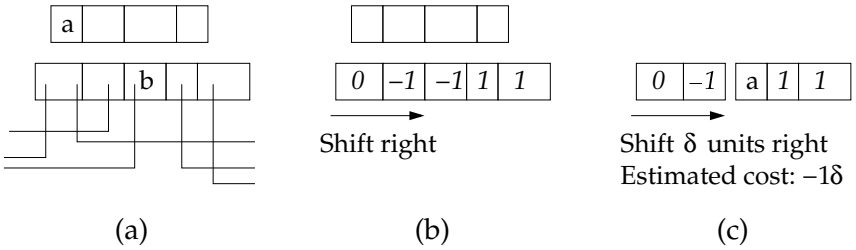


Figure 3.9: *The "prefer" method for estimating netlength change when moving a cell from one row to another. (a) We wish to test the swap of modules a and b. The connections in the second row are shown. (b) The prefer values for each module. The prefer value is the number of nets for which this module is left-extreme minus the number of nets for which this module is right-extreme. (c) If shifting the row δ units to accommodate for the size of a does not change which modules are extreme the new netlength is δW_s when W_s is the sum of prefer values on the left. Otherwise this is a rough estimate.*

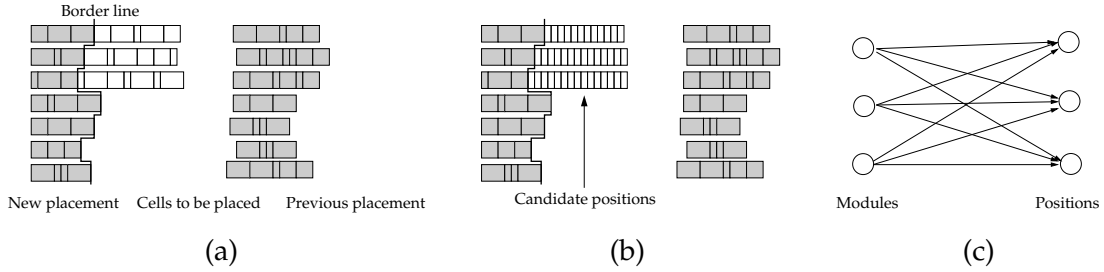


Figure 3.10: *The Domino placement of a region. (a) Shaded cells left of the border are fixed according to their new position. Shaded cells on the right are fixed according to their old position. The clear cells are from the region to be placed. (b) Candidate positions for sub-cells are constructed from the rows. (c) A transportation problem is formulated. The cost of the edges is the estimated netlength contribution of each sub-cell. Supply from each module is the number of its sub-cells. Demand in each location is one sub-cell.*

size and then a set of legal positions of the sub-cells is generated in each row. A transportation problem of sub-cells is now formulated. The cost of moving a sub-cell to a location is estimated based on netlength and the problem is solved. Doll et al. discovered that most of the sub-cells end up next to each other. Therefore each cell is placed at the position of most of its sub-cells. Due to the estimated netlength and the size of the transportation problem Doll et al. limit the number of modules in each region to between 20 and 40. Because Domino places modules according to a transportation problem and repeatedly legalizes a relaxed placement it has the ability to escape local minima of e.g. greedy swapping based heuristics. Figure 3.10 gives an overview of Domino's placement procedure. The Domino netlength approximation scheme is elaborated on in [16].

In [89] the global solution is considered so good that instead of using the simulated annealing final placement the authors opt for a greedy approach. For each module the

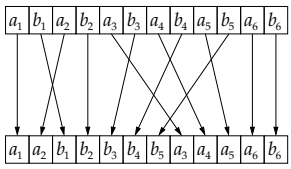


Figure 3.11: *The Mongrel placement improvement. The sequences* $< a_1, \ldots, a_6 >$ *and* $< b_1, \ldots, b_6 >$ *are interleaved optimally by dynamic programming.*

complete neighborhood is searched for the best swap. The neighborhood is relative small; only cells a few neighbors away are considered as candidates for swapping.

Vygen [86, 88] consider rows of the final placement. He uses the term "zone" for a maximal part of a row which is not intersected by a fixed object. The initial assignment is done based on the global placement as described in 3.1.3. Now some zones may contain too many modules and some too few. To overcome this problem he solves a minimum-cost network flow problem where overloaded zones are sources and zones with free capacity are sinks. The cost function is based on an estimate of moving a unit (e.g. an inverter not a module) from one zone to another. After the minimum-cost flow problem has been solved he solves a knapsack problem to actually determine which modules to move.

A very interesting idea is by Hur and Lillis [39]. Their placer Mongrel considers only movement within rows. Let a row consist of the modules W. Now a subsequence $A = a_1, \ldots, a_n$ of the modules of W is extracted and a subsequence $B = b_1, \ldots, b_m = W \setminus A$ is also extracted. They now preserve the relative order of the modules in each of the subsequences but to improve the placement they consider an optimal interleaving of the subsequences (e.g. $a_1, a_2, b_1, a_3, b_2, b_3, a_4, \ldots$). It turns out that determining the optimal interleaving can be done by dynamic programming in time $O(mn + p(n+m))$ where p is the total number of pins on the modules. The idea is illustrated on figure 3.11

Another novel idea by Hur and Lillis [39] is to cluster the rows. During the final placement they create clusters of between 3 and 10 modules. Now the interleaving method described above is used on the clusters.

3.2.3 Guided Local Search

The guided local search based method of Færø et al. [21] is interesting because of its simplicity and generality to modules of arbitrary size.

The local search is an iterative process. In each iteration an optimal – with respect to the objective function – either horizontal or vertical position of a module m is determined. The local search method can determine this position in $O(|\mathcal{P}_m| + (|\overline{\mathcal{M}}_m| + |\mathcal{M}_m| + |\mathcal{N}_m|) \log(|\overline{\mathcal{M}}_m| + |\mathcal{M}_m| + |\mathcal{N}_m|))$ time where $\mathcal{P}_m$ is the number of pins of m, $\mathcal{M}_m$ is the number of modules directly connected to m, $\mathcal{N}_m$ is the number of nets connected to m and $\overline{\mathcal{M}}_m$ is the number of modules which overlaps horizontally or vertically with m [6]. To move towards legal placements overlap between pairs of modules is penalized in the objective function. In order to reduce the search space a region based search is used for large circuits.

The local search is controlled by the Guided Local Search (GLS) meta-heuristic (see e.g. [85]) which has been quite successful for other problems. GLS enables the local search to escape local optimum by perturbing the penalized overlap part of the function. Also the Fast Local Search methodology, which is part of a standard GLS framework, limits the number of candidate modules for local search.

The primary problem with the GLS method seems to be that in order to ensure legal placements the overlap penalty may be set very high making it more difficult to move cells. In other words the netlength and overlap penalty are contradicting objectives. Also the local search may be too slow for very large circuits even with region based search. However GLS does produce good results.

In [25] the GLS method was parallelized and also used with a congestion objective.

3.3 Comparison of Placement Heuristics

Results of the standard-cell benchmarks which are described in section 5 have been reported for most of the previously described placement heuristics by their authors. However the results were investigated by Madden [57] who has been in contact with many of the authors. Madden discovered that there were several subtleties in comparing the standard-cell methods.

The biggest problem with the results is that the following items are not specified in the circuit description:

- **Row height** The benchmarks are standard-cell but the row height is not specified and authors have interpreted the row height freely. Some placement heuristics operate with row spacing equal to cell height while others completely disregard row spacing.
- **Dimensions** The dimensions of the layout area are not specified and heuristics use different number of standard-cell rows.

[6] Two modules overlap horizontally if their horizontal extends overlap

- **Pad positions** The positions of the IO-pins of the circuits are not specified so some placement heuristics may be more lucky placing the pads than others. More importantly however is that IO-pins in some cases are positioned on the border of the placement area and in other cases some distance from it.

Another problem is pin location on the modules. Although all the heuristics measure wire-length using bounding-box netlength some disregard the pin offsets and use the module centers or lower left corner.

Run time comparisons are also difficult. Firstly the machines used for benchmarking may be different. Secondly some heuristics use random starts and often only the best achieved result and its sole running time is reported. Thirdly most heuristics have a number of different parameters, e.g. constants for simulated annealing or overlap penalties. Fine-tuning these parameters may change the quality of the heuristic. Finally many of the global placement methods use some form of legalizer and final placer with only the end-result and running time reported.

Although these issues makes it hard to compare the heuristics we enlist the results from various articles as they were originally presented by their authors in table 3.1. A similar table was presented by Madden [57] but we have omitted some of the placement heuristics and added the GLS based heuristic of Færø et al.

3.4 Minimizing Area and Topological Structures

Area minimization is the other popular objective. It makes little sense to do area minimization of standard cells and we will not consider this matter here. Instead we focus solely on general-cells. The reason to consider area minimization is because the area minimization algorithms are often connected with topological structures. We will use a topological structure in our legalization algorithm and therefore the right choice of structure is important.

Topological structures describe the relative position of the modules (floor-plan). The set of slicing floor-plans are floor-plans which can be achieved by recursively bisecting the placement area (see figure 3.12). The non-slicing floor-plans are floor-plans that are not slicing. Slicing floor-plans can be represented by a binary tree whereas general floor-plans (slicing and non-slicing) are more complicated to represent. We will only look at representations for general floor-plans here.

3.4.1 Sequence-Pair

The sequence-pair model is by far the most researched model for area minimization and macro-block placement. Since the sequence-pair model is the primary subject of section 4 we will not go into details of the representation here.

Circuit	Eisenmann et al. [19]	FAR$^{(I)}$ [37]	ARP [20]	NRG[76]	Dragon [89]
industry1	-	-	1.50/376	-	-
industry2	14.6 / 1284	14.4/213	-	-	12.9/1461
industry3	45.1 / 1605	42.7/131	48.1/4253	-	42.3/2849
avqlarge	5.38 / 2032	8.25/277	7.16/11202	-	5.25/1984
avqsmall	4.91 / 1741	7.40/527	6.42/8534	-	5.17/1420
biomed	1.78/284	4.23/31	1.83/1290	-	-
golem3	-	-	-	-	77.6/8422
struct	0.338/40	-	0.34/116	0.315/-	-
primary1	0.870/37	-	1.08/95	0.900/-	-
primary2	3.72/152	3.45/24.6	4.02/504	3.41/-	-
CPU used	Alphastation 250/4-266	Pentium 3/ 850 Mhz	Sun Ultra/ 140	Sun Sparc/ 20	PC$^{(II)}$ 500 Mhz
Circuit	TimberWolf 7 [80]	Mongrel$^{(III)}$[39]	Itools$^{(IV)}$	XQ$^{(V)}$	GLS [21] $^{(VI)}$
industry1	-	-	-	1.75/90	1.86 $\rightarrow$ 1.63
industry2	15.8/9252	12.45/3443	11.4/-	16.9/1.14	14.46 $\rightarrow$ 14.3
industry3	44.9/8766	37.90/4814	39.6/-	48.7/1260	42.7 $\rightarrow$ 42.6
avqlarge	7.16/21086	5.00/8344	4.78/-	8.55/1320	6.88 $\rightarrow$ 6.79
avqsmall	6.42/17.44	4.62/8222	4.48/-	-	-
biomed	3.96/2640	1.48/480	2.90/-	4.32/1254	3.47 $\rightarrow$ 3.44
golem3	-	-	79.9/-	119/8400	118 $\rightarrow$ 114
struct	-	0.278/90	0.272/-	0.77/54	0.778 $\rightarrow$ 0.744
primary1	1.08/168	0.87/162	0.799/-	1.14/31	0.987 $\rightarrow$ 0.949
primary2	4.02/922	3.13/249	3.37/-	4.10/150	3.64 $\rightarrow$ 3.61
CPU used	DEC Station 5000/200	Pentium 2 400 Mhz	-	Pentium 2 500 Mhz	Pentium 3 800 Mhz

Table 3.1: *Comparison of the standard-cell benchmarks. The numbers are 'wire-length (meters)/run time (seconds)'. '-' indicates number missing from original article.* $^{(I)}$ *The results for FAR are after legalization and simple optimization.* $^{(II)}$ *CPU brand not reported but is likely to be Pentium 2.* $^{(III)}$ *Average results from article.* $^{(IV)}$ *ITools is the commercial version of TimberWolf. We have found no details on its algorithm and the results are from [57].* $^{(V)}$ *According to [24] XQ uses the algorithm of Vygen and the results are from [24].* $^{(VI)}$ *GLS was conducted on TimberWolf initial solutions. All runs of GLS lasted 60 minutes. Results are reported as Timberwolf Solution* $\rightarrow$ *GLS-solution. The circuits are elaborated on in section 5.*

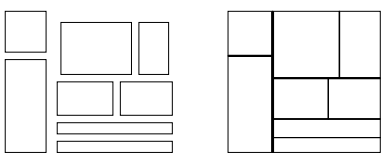


Figure 3.12: *Slicing floor-plan. The placement on the right is converted to a floor-plan on the left. The placement is achieved by recursive bisection. The thinner the line the deeper the recursion step.*

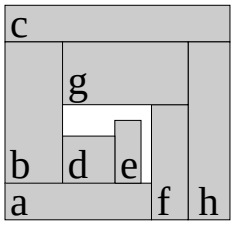


Figure 3.13: *Macro-block example containing 8 modules.*

The model was introduced in 1996 by Murata et al. [63]. The primary idea of the model is to describe relations between any pair of modules by two sequences of module references. The intermodule relation is simple to determine from the sequences. If module a precedes module b in the second sequence and module a precedes module b in the first sequence then a is left of b. Similarly if a precedes module b in the second sequence and module b precedes module a in the first sequence then a is below b. Based on this description the placement of figure 3.13 corresponds to the sequence pair:

$$\langle < c, b, g, d, e, a, f, h >, < a, b, d, e, f, g, h, c > \rangle$$

Converting the two sequences into a placement has been the topic of extensive research and we will elaborate on it in section 4.4 however we will present running times of the algorithms. The original method of Murata et al. had running time $O(n^2)$ for placing n blocks. Tang et al. [81] improved this $O(n \log n)$ and later to $O(n \log \log n)$ [82].

Pisinger [72] also improved the placement algorithm in 2002. Here the placement is done with a very simple packing envelope structure in combination with a fast priority queue data-structure and his algorithm also has running time $O(n \log \log n)$. The resulting placements are semi-normalized [7] and are therefore not completely true to the the sequence-pair representation. However it is proven in the article that an optimal sequence-pair packing must be semi-normalized[8]. The improvement of Pisinger lies both in the fact that the placement algorithm is fast and in the fact that it produces more compact placements than the true sequence-pair based algorithms.

Common for the sequence-pair algorithms is that they were implemented in a simulated annealing framework. Results of the algorithms are in table 3.2.

Various authors have contributed to the sequence-pair paradigm. In [93, 45, 26] general rectilinear blocks are considered. Fixed module and obstacles are considered in [62, 82], and other topics are discussed in [55, 53, 5, 64].

[7] In a normalized placement all modules are moved as far left and down as possible

[8] Optimal in the sense that the area is minimized

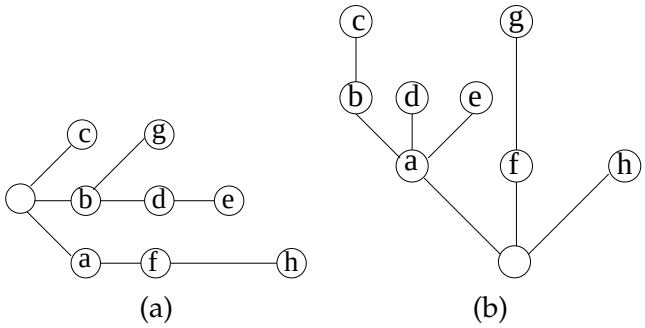


Figure 3.14: *O-trees of macro-block example from figure 3.13. (a) Horizontal O-tree. (b) Vertical O-tree.*

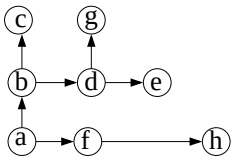


Figure 3.15: *The B*-tree of the placement of figure 3.13. Module a is the root of the tree.*

3.4.2 O-tree and B*-tree

The O-tree structure was introduced in [31] by Guo et al. in 1999. The representation uses less space, transformation to placement takes time $O(n)$ and placements are more compact than those of the *original* sequence-pair algorithm. The O-tree structure is a single-source shortest-path tree in *either* a horizontal or vertical constraint graph. Nodes in the tree corresponds to modules and children's position are determined by their parent, e.g. they cannot be moved further left (or down) than the right (or upper) edge of the parent. The root node corresponds to the left or lower edge of the chip (see figure 3.14). The placement-algorithm uses a tree-traversal and an envelope structure. The authors implemented three placement heuristics; a constructive algorithm, a deterministic algorithm for refinement and an algorithm based on random O-trees.Table 3.2 shows results of their deterministic algorithm. Rectilinear block placement is considered in [70].

The B*-tree by Chung et al. [11] is similar to the O-tree but consist of a binary tree. If a node j is the left child of node i then module j is adjacent and right of module i. If j is the right child of i then j is above i and has same x-coordinate as i. The placement algorithm is similar to that of the O-tree and has linear time complexity. In addition the authors describe how to extend the algorithm to handle pre-placed, rectilinear and *soft* modules – modules with specific area but unspecified dimensions. The experimental results are shown in table 3.2. Figure 3.15 shows a B*-tree for the modules of figure 3.13.

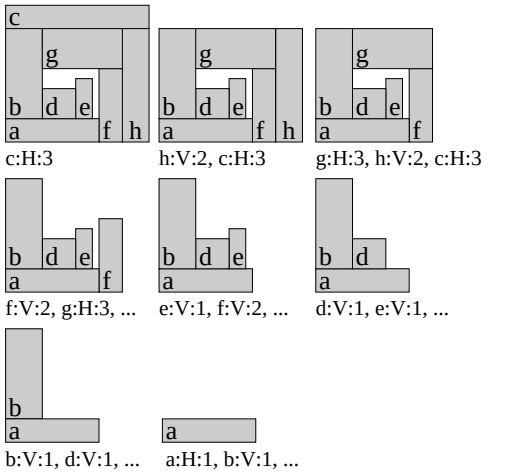


Figure 3.16: *CBL construction from macro-cell example of figure 3.13.The CBL is constructed backwards. In each step a block is deleted. Deletion is categorized as horizontal if it "frees" most blocks in the horizontal direction and vertical if it "frees" most blocks in the vertical direction. The number of blocks with incident edges to the deleted block in the corresponding constraint graph is recorded.*

3.4.3 Corner Block List

In 2000 Hong et al. introduced the Corner Block List in [36]. The Corner Block List (CBL) is a topological structure for a subset of floor-plans called mosaic floor-plans. The set of mosaic floor-plans is a superset of slicing floor-plans but a strict subset of non-slicing floor-plans. The CBL structure is a bit-pattern describing how each block is to be placed relative to previous blocks. Constructing a corner block list from a placement is simple. First the top-right module is deleted from the placement. The deletion is categorized as either a horizontal or a vertical deletion depending on whether removing the module reveals covered modules below or to the left. The CBL construction is demonstrated in figure 3.16. A reference to the module, deletion type and number of revealed modules are stored in front of the CBL. Converting the CBL to a placement can be done in $O(n)$ time. The CBL placement algorithm was combined in a simulated annealing framework. The neighborhood of the local search does not always lead to legal floor-plans, but the authors discard illegal floor-plans.

Because of the floor-plan-nature of CBL-placements the CBL's main strength lies in its ability to handle soft modules. However the authors comment little on this matter other than to give results for soft placements.

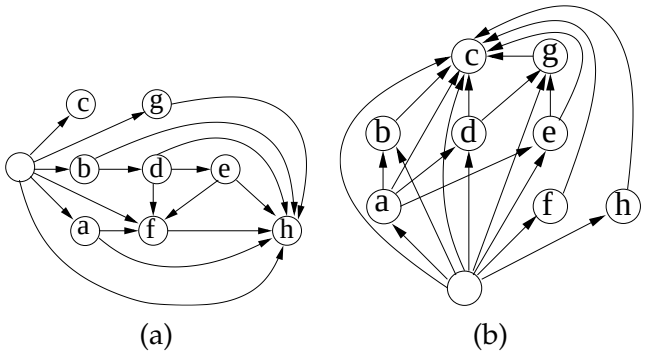


Figure 3.17: *TCG of placement from figure 3.13. The figures illustrate the complexity of the TCG.*

3.4.4 BSG

In 1996 Nakatake et al. introduced the Bounded Sliceline Grid (BSG) structure [66]. The BSG is described as an infinite grid consisting of rooms, which defines a horizontal and a vertical graph. In either graph exactly one edge crosses each room. For the placement each module is assigned to one room but not all rooms need to be filled. The placement is constructed from a longest path algorithm and has quadratic running time. The advantages of the BSG are unclear however the authors show that their algorithm is capable of packing in general shapes by preassigning dummy modules to rooms where modules are not allowed to be placed.

3.4.5 Transitive Closure Graph

Lin and Chang [56] proposed the transitive closure graph (TCG) for modeling non-slicing floor-plans. The TCG-model consists of a horizontal and a vertical dependency graph where each node corresponds to a module. An edge from i to module j in the horizontal graph means that j must be to the right of i, while a similar edge in the vertical graph means that j must be above i. Transitive edges are included in the graphs which makes moves of modules in the graphs simple. The TCG of the placement from figure 3.13 is shown in figure 3.17. Transforming the TCG to a placement is with a longest path algorithm and has a quadratic running time. However the authors present fine results with running times matching CBL and Fast-SP. The results are shown in table 3.2.

3.4.6 Comparison of Topological Structures

The quality of the various topological structures and the time spend on the results are listed in 3.2. Although some of the results are quite impressive one should of course note that the results from the simulated annealing based algorithms in general are

Algorithm	apte	xerox	hp	ami33	ami49	Processor
Original sequence-pair					44.89 / 1881.6	SunIPX
Fast-SP	46.92 / 1	19.80 / 14	8.947 / 6	1.205 / 20	36.50 / 31	166 Mhz. Sun Sparc Ultra-1
Semi-normalized SP	46.92/0.80	19.80/1.10	8.947/1.01	1.178/4.96	36.33/9.33	Pentium-III 1 Ghz
Iterative O-tree	63.3 / 0.65	25.9 / 0.44	14.3 / 0.26	1.69 / 2.83	54.6 / 11.2	200 Mhz. Sun Sparc Ultra-1
Iterative B*-tree	46.92 / 0.11	20.06 / 0.39	9.17 / 0.21	1.27 / 7.83	37.43 / 39.24	200 Mhz. Sun Sparc Ultra-I
SA B*-tree	46.92 / 7	19.83 / 25	8.95 /55	1.27 / 3417	36.80 / 4752	200 Mhz. Sun Sparc Ultra-I
Corner block list	-	20.96 / 30	66.14 / 32	1.201 / 36	38.58 / 65	Sun Sparc 20
TCG	46.92 / 1	19.83 / 18	8.947 / 20	1.20 / 306	36.77 / 434	433 Mhz. Sun Sparc Ultra 60

Table 3.2: Comparison of macro-cell placement algorithms (area mm^2/time sec). The benchmarks (apte, xerox, hp, ami33, and ami49) are elaborated on in section 5.

the *best* results – not the average result of a number of runs. This makes comparison difficult since good results may be a matter of random seed choice.

3.5 Branch-and-Bound Algorithm

We have only found one exact branch-and-bound algorithm for the placement problem. In 1991 Onodera et al. [68] published a very simple branch-and-bound algorithm. The branch-and-bound algorithm introduces block constraints of the form "left of", "above" etc. in the search tree and prunes if optimum of the resulting linear-programming formulation is worse than the current best, if the introduced constraint is redundant or if the current placement violates some aspect ratio constraint. The algorithm is only capable of placing less than six modules and for more than six modules a clustering method is used.

It is interesting that very little work has been done in terms of area-minimized rectangle packing. There exists branch and bound algorithms for rectangle bin packing and strip packing which are quite efficient [59, 22]. The problem is likely discovering good bounds for area-minimization.

Also in terms of minimizing netlength an exact branch-and-bound algorithm makes little sense when the netlength is based on some rough estimate.

3.6 Legalization

Some of the placement methods of the previous sections do not result in a legal placement. Therefore a legalization step must be introduced.

3.6.1 Simple Legalization Strategies

A legalization strategy often exerted by authors of standard-cell placements is to simply sort the modules according to x- and y-coordinate and place the modules in rows using the y-coordinate to determine row number and the x-coordinate to determine module order of that row. This method is used in e.g. [49],

3.6.2 Guided Local Search

The guided local search method described in section 3.2.3 can also be used strictly to legalize a placement [24]. This is done simply by setting the penalty for overlap sufficiently large, and even large circuits can be legalized in a matter of minutes without changing the netlength of a *global placement* more than 10-20 percent.

3.6.3 Overlap Removal by Sequence-Pair

In [65] Nag and Chaudhary describe a method to remove overlap using the sequence-pair representation (see section 3.4.1). The current overlapping placement is converted to a sequence-pair representation. The sequence-pair representation is converted to the horizontal and vertical constraint graphs of the original sequence-pair model (which will be described in section 4.4.1). An upper limit is imposed on the horizontal and vertical dimensions of the chip. This in turn corresponds to an upper limit on the longest path in either constraint-graph. Now the most violating path in either graph is determined. The violating path is then searched for an edge-swap. The edge-swap procedure roughly converts a horizontal module relation to a vertical one or vice versa. The procedure reduces overlap in the direction (horizontal or vertical) of the longest path, but with the risk of introducing new overlap in the opposite direction. The path-search determines the best edge-swap. The algorithm proceeds iteratively reducing overlap and stop when a feasible placement is reached. The original placement of the modules is taking into consideration by adding a distance-weight to each arc in the constraint graphs. When a feasible placement has been determined the surplus space between modules is divided among the arcs using the original distances as a reference. This ensures that the final feasible placement is close to the original placement. The algorithm seems fairly slow compared to e.g. GLS (see section 3.2.3). Only one experiment is reported. The experiment removes 9 overlaps within 2 minutes on an UltraSparc2.

3.7 Post Optimization

When the relative order of the modules has been determined by final placement there is often room for improvement.

Vygen [88] use the order of the modules at the end of the final placement to construct a linear program which minimizes the bounding-box netlength of the modules but with the ordering preserved. This turns out to be the dual of a minimum-cost-flow problem (see appendix C.2.3) and can be solved in quadratic time which unfortunately is too expensive. Therefore he uses the algorithm on small regions of the placement area.

For minimizing the bounding-box netlength of a *single* row with fixed ordering Brenner and Vygen [6] improve an algorithm by Kahng et al. [44]. Brenner and Vy-

gen's algorithm runs in time $O(m \log \log m)$ for the cases with unweighted nets and $O(m(\log m)^2)$ for the weighted case. However the method is too complicated to describe here.

4 Sequence-Pair Legalization

In this section we present a legalization algorithm based on the sequence-pair representation which was briefly presented in section 3.4.1. The algorithm has running time $O(n \log n)$ for n rectangles and consists of two large steps.

1. The current possibly overlapping placement will be converted to a sequence-pair.
2. The sequence-pair representation is used to construct a new non-overlapping placement.

Before considering these two steps of the legalization we will explain the sequence-pair representation in more detail than was given previously in section 3.4.1. After this brief introduction we will consider conversion to and from sequence-pair.

Why Sequence-Pair We have chosen the sequence-pair representation as topological representation for several reasons. Firstly the sequence-pair representation is by far the most researched of the representations. Secondly sequence-pairs consists of codes rather than complicated tree or graph structures. This makes it easy to represent and build the abstract representation. Thirdly the sequence-pair representation contains information of relative placement of any two rectangles. It is harder to extract relative placement of two arbitrary rectangles in e.g. the B*-tree representation.

4.1 Packing Problems

The sequence-pair representation is used for packing problems and therefore we will avert to these for this entire section. We begin with the following definitions:

The first definition corresponds to the one of section 2.2.4.

Definition 4.1. Placement (of rectangles) Let R be a set of rectangles with width $r_w \in \mathbb{N}$ and height $r_h \in \mathbb{N}$ for $r \in R$. Then a placement $P = (x, y)$ of R is a pair of maps $x : R \to \mathbb{N}_0$ and $y : R \to \mathbb{N}_0$ which gives the coordinates of each rectangle.

Definition 4.2. Packing Let R be a set of rectangles with width $r_w \in \mathbb{N}$ and height $r_h \in \mathbb{N}$ for $r \in R$. Then a packing $P = (x, y)$ of R, $x : R \to \mathbb{N}$ and $y : R \to \mathbb{N}$, with width $w \in \mathbb{N}$ and height $h \in \mathbb{N}$ is a placement of R such that no two rectangles overlap and all rectangles are contained within the rectangular region $[0, w] \times [0, h]$. I.e. $\forall r, s \in R : \ (x(r)+r_w \leq x(s)) \vee (x(s)+s_w \leq x(r)) \vee (y(r)+r_h \leq y(s)) \vee (y(s)+s_h \leq y(r))$ and $\forall r \in R : \ x(r) \geq 0 \wedge y(r) \geq 0 \wedge x(r) + r_w \leq w \wedge y(r) + r_h \leq h$.

Note that our definition requires that a packing is a *legal* placement in the sense that no two rectangles overlap.

We also define rectangle relations:

Definition 4.3. Left, right, above and below For two rectangles r and s in a placement $P = (x, y)$ we say that r is left of s if and only if $x(r) + r_w < x(s)$. Similarly we define right ($x(r) > x(s) + s_w$), below ($y(r) + r_h < y(s)$) and above ($y(r) > y(s) + s_h$) relations.

Definition 4.4. RPDP The rectangular packing decision problem (RPDP) of the instance (R, w, h) is to determine if there exists a packing of the rectangles R with width $w \in \mathbb{N}$ and height $h \in \mathbb{N}$.

The RPDP is a decision problem and easily shown to be NP-complete by reducing from 2-partition.

Definition 4.5. ARPP The area rectangular packing problem (ARPP) of the instance (R, a) is to determine if there exists a packing of the rectangles R with area a such that the width w and height w of the packing is $a = w \cdot h$.

Although not immediately obvious RPP is polynomially reducible to ARPDP (see e.g. [63]), and so ARPP is also seen to be NP-complete.

Definition 4.6. Minimal Area Rectangle Packing Problem (MARPP) The MARPP of a set of rectangles R is to determine the minimal $a = w \cdot h \in \mathbb{N}$ such that there exists a packing P of (R, w, h).

Note that MARPP is of course NP-hard.

4.2 The Sequence-Pair Representation

The original sequence-pair representation was introduced by Murata et al. [63]. The purpose of this abstract representation of legal placements was to use it for solving MARPPs.

4.2.1 Gridding

To explain the sequence-pair representation we will first demonstrate a simple technique to transform a placement to a sequence-pair by hand. This was also the way it was presented by Murata et al.

Figure 4.1 (a) illustrates 6 rectangles. Their relative placement is converted to a sequence-pair as follows:

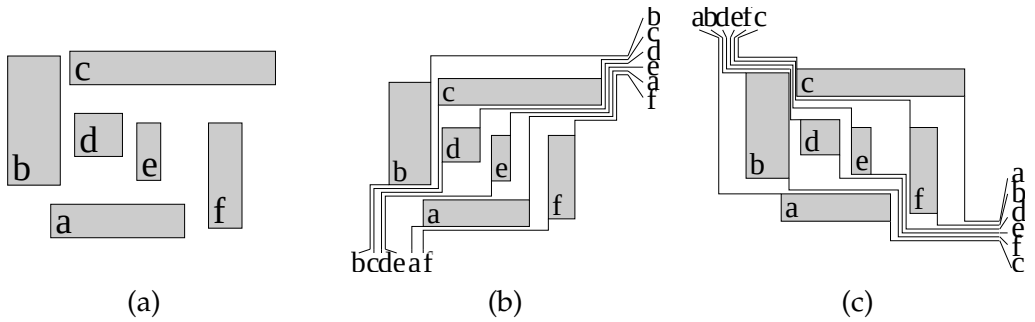


Figure 4.1: *Construction of a sequence-pair using gridding. (a) The placement of the rectangles. (b) positive step-lines. (c) negative step-lines. The rectangle sequence arising from positive step-lines is* $< b, c, d, e, a, f >$. *The sequence from the negative step-lines is* $< a, b, d, e, f, c >$.

- For each rectangle draw alternating leftwards horizontal and downwards vertical lines from its lower left corner until the the lower left corner of the rectangle placement area is reached. Do not cross other rectangles or their step lines.
- Do the same from the upper right corner of each rectangle to the upper right corner of the placement area with rightwards and upwards orthogonal lines and do not cross any of the lines from the previous step.

These lines are called positive step-lines. We may order the lines from left to right at e.g. the lower left corner or from top to bottom at the upper left corner (see figure 4.1(b)). Because the lines do not cross the order is maintained throughout the placement area and it allows us to construct a sequence of rectangles. Call this sequence $\mathbf{A}$.

Similarly we construct a sequence, $\mathbf{B}$, by drawing lines from upper left corners and lower right corners of the rectangles to respectively the upper left corner and lower right corner of the placement area (figure 4.1(c)). These lines are called negative step-lines. Combined the two sequences form the *sequence-pair* $\langle \mathbf{A}, \mathbf{B} \rangle$.

4.2.2 Properties of the Sequence-Pair

Based on the gridding process we can conclude a nice property of the sequence-pair $\langle \mathbf{A}, \mathbf{B} \rangle$.

Theorem 4.1. *(Murata et al. [63]) Assume* $\langle \mathbf{A}, \mathbf{B} \rangle$ *is a sequence-pair of a packing* P *produced by gridding.*

If a *precedes* b *in sequence* $\mathbf{A}$ *(i.e.* $\mathbf{A} =< \ldots, a, \ldots, b, \ldots >$*) and* a *precedes* b *in sequence* $\mathbf{B}$ *(i.e.* $\mathbf{B} =< \ldots, a, \ldots, b, \ldots >$*) then* a *is left of* b *in* P.

Similarly if b *precedes* a *(i.e.* $\mathbf{A} =< \ldots, b, \ldots, a, \ldots >$*) in sequence* $\mathbf{A}$ *and* a *precedes* b *in* $\mathbf{B}$ *then* a *is below* b *in* P.

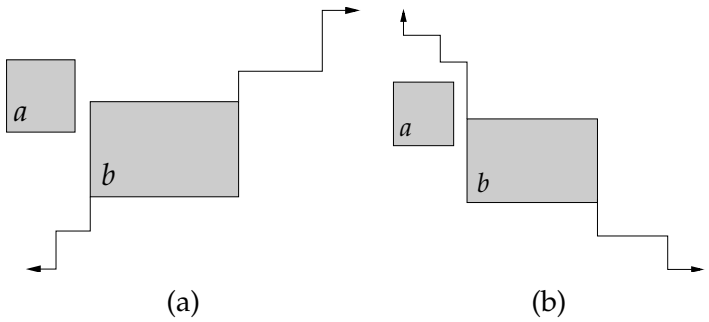


Figure 4.2: *Illustration for the proof of theorem 4.1. Assume a precedes b in both sequences of the sequence-pair. (a) The positive step-lines require that a be left of or above b. (b) The negative step-lines require that a be left of or below b.*

Proof. The proof is based on [63].

Assume a precedes b in both sequences. When we drew the step-lines for b, a must have preceded b in the ordering of both the positive and negative step-lines. Since we drew the positive step-lines from the top-right corner of b to the top-right corner of the placement area, and from the lower-left corner of b to the lower-left corner of the placement area with horizontal and vertical step-lines, a must be left or above b (see figure 4.2(a)). Similarly a must have preceded b in the ordering of the negative step-lines and therefore a must be left of or below b (see figure 4.2(b)). From this we conclude that a must be left of b.

The other case is proven similarly. □

Note of course that the theorem is symmetric in the sense that if a proceeds b in both sequences then a is right of b and if a proceeds b in the **A**-sequence and precedes b in the **B**-sequence then a is below b.

P-admissible As described in the previous paragraphs a sequence-pair is a code of a packing. All sequence-pairs for a set of rectangles constitute a solution space and Murata et al. proposed the following 4 properties of a "good" solution space of codes which could be used for combinatorial search:

1. The solution space is finite
2. Every code is feasible
3. Realization of a code is possible in polynomial time.
4. There exists a code which corresponds to an optimal solution of the problem.

They call such a solution space *P-admissible*. The purpose of these requirements should be clear; that the solution space can be searched in finite time, without considering infeasible solutions and the optimal solution is part of the solution space. Apart from item 4 it will become clear from the following sections that the sequence-pair representation is P-admissible. To see 4 keep in mind that in a optimal solution to the *Minimal Area Rectangle Packing Problem* any rectangle can be moved as far left and down as possible until it abuts another rectangle. It should also be noted here that the sequence-pair representation is not P-admissible in terms of the placement problem since there exists solutions to the placement problem in which rectangles cannot be moved left and down without increasing netlength (see e.g. 2.3).

4.3 From a Placement to Sequence-Pair

In this section we will present an algorithm which can convert a placement into a corresponding sequence-pair. In the previous section we explained the sequence-pair representation and used gridding to decide the ordering of the two sequences. This method is not in strict sense an algorithm.

Imahori et al. [41] presented an algorithm which runs in time $O(n \log n)$ but this algorithm does not consider possible overlap. Originally unaware of Imahoris algorithm we have created a different algorithm which also has running time $O(n \log n)$. Our algorithm is based on gridding. The algorithm is a combination of two algorithms. One algorithm clears overlap by reducing the dimensions of the rectangles while another algorithm converts the non-overlapping packing to a sequence-pair.

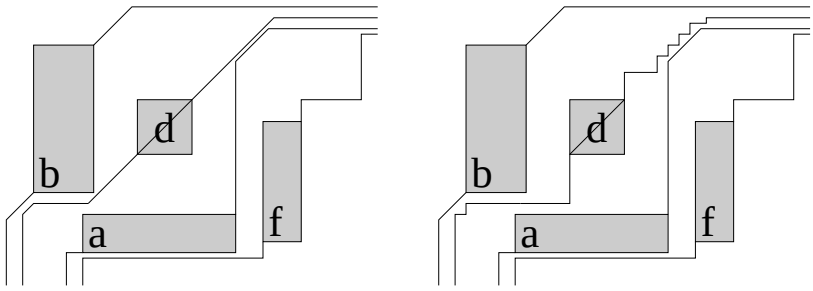
First we will describe a very fast heuristic which does not always produce the correct sequence-pair. This method also has running time $O(n \log n)$ but the constants are smaller since it is based completely on sorting.

We can justify the $O(n \log n)$ running times by this theorem.

Theorem 4.2. *No correct conversion from a placement to a sequence-pair can be done faster than sorting a set of distinct integers.*

Proof. Assume $S \subset \mathbb{Z}$ is a set of distinct integers. For each number $i \in S$ create a rectangle with coordinates $(x, y) = (i, 0)$ and dimensions $(w, h) = (1, 1)$. Then the $\mathbf{A}$ (and $\mathbf{B}$) sequence of the packing corresponds to a sorted sequence of the numbers. This is true since according to theorem 4.1 a block a is left of another block b if and only if a precedes b in both sequences of the sequence-pair. Further a is left of b if and only if the number corresponding to a is less than the number corresponding to b. □

The same argument can be used for other abstract representations (e.g. O-trees and B*-trees).



(a) (b)

Figure 4.3: *(a) Gridding with diagonal line fragments. (b) The diagonal line fragments of* $\mathbf{d}$ *can be converted to straight line fragments thus obeying the rules of gridding.*

4.3.1 A Heuristic Approach

The purpose of transforming a packing to a sequence-pair is to legalize the placement fast. We will begin with a simple heuristic for two reasons.

1. The heuristic method forms the base of the algorithm which we will present in the following section.
2. The heuristic method is several times faster than the algorithm allowing speed-ups whenever an approximate sequence-pair suffices.

First we note that allowing diagonal line fragments during the gridding phase is absolutely legal as long as no lines cross. The argument supporting this is that diagonal lines can be converted to a stair case of sufficiently small steps of horizontal and vertical lines thus obeying the rules of the sequence-pair gridding (see figure 4.3).

The idea of the heuristic is simple. It is based on the assumption that all rectangles are points (e.g. the center of the rectangle). Allowing diagonal line fragments and assuming that rectangles are points we get a gridding as shown on figure 4.4. If all diagonal lines have equal slope they do not intersect and we can order them as we previously ordered the step-lines. Simply ordering the rectangles according to their diagonals gives us the two sequences. The method is illustrated on figure 4.5

Since the order of the diagonals is maintained throughout the plane, the order can be determined by the intersection point of the diagonal and any horizontal line. In the following we choose the x-axis (horizontal line with $y = 0$).

The equation of a diagonal through a point $\mathbf{p} = (p_x, p_y)^t$ is

$$x + \alpha \cdot y - p_x - \alpha \cdot p_y = 0 \tag{4.1}$$

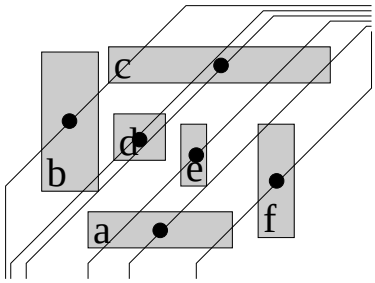


Figure 4.4: *If rectangles are considered as points and we allow diagonal grid lines we get this gridding of the packing from figure 4.1.*

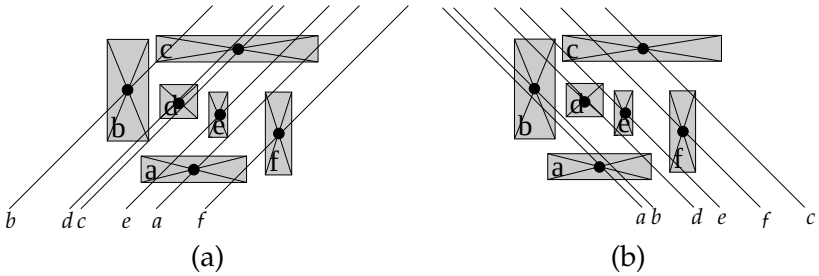


Figure 4.5: *Extending the line-segments through the rectangles to the bottom of the packing area we can get the sequence-pair by looking at the intersection of the diagonal with the bottom. (a) is the* $\mathbf{A}$ *sequence. (b) is the* $\mathbf{B}$ *sequence. The sequence-pair is* $\langle < b, d, c, e, a, f >, < a, b, d, e, f, c > \rangle$*. Notice that the only difference from this and the sequence-pair of figure 4.1 is that* d *and* c *have swapped place in the first sequence. This has the unfortunate effect of requiring that* c *be placed to the right of* d *instead of above.*

Where α is a constant which describe the slope of the diagonal. Intersection with the horizontal line $y = 0$ occurs at $x = p_x + \alpha p_y$. Positive α gives $\mathbf{A}$-sequence diagonals while negative α gives $\mathbf{B}$-sequence diagonals.

The previous discussing allow us to define algorithm (heuristic) 4.1 for determining the sequence-pair of a packing.

Algorithm 4.1: Heuristic for sequence-pair encoding

Input(*A placement* $P = (x, y)$ *of rectangles* R *with* center-*coordinates* $(x(r) + \frac{r_w}{2}, y(r) + \frac{r_h}{2})^t$ $r \in R$, *and real number* α) ;
$\mathbf{A} = [$ R sorted ascending according to $x(r) + \frac{r_w}{2} + \alpha(y(r) + \frac{r_h}{2})]$;
$\mathbf{B} = [$ R sorted ascending according to $x(r) + \frac{r_w}{2} - \alpha(y(r) + \frac{r_h}{2})]$;
return *sequence-pair* $< \mathbf{A}, \mathbf{B} >$

Notice on figure 4.5 that even in the illustrated simple case the heuristic does not produce the correct result. The error comes from assuming that all modules are points. The method performs best with instances of rectangles of more or less equal size.

4.3.2 A Sweep-Line Algorithm

Using the heuristic of the previous section we can construct a simple sweep-line algorithm for creating the sequence-pair of a placement. Once again we use the gridding methodology and once again we use diagonals as opposed to step lines. The algorithm does not allow for overlap in the placement. We only describe the algorithm for the $\mathbf{A}$-sequence since the $\mathbf{B}$-sequence can be constructed similarly.

Instead of drawing diagonals through the center of the rectangles from bottom to top of the packing area we only draw diagonals from the upper right corner of rectangles. Secondly we add line-segments from lower left to upper right corner of each rectangle. We call these line-segments *rectangle-segments*. This is shown on figure 4.6(a).

According to the gridding rules no "step-lines" are allowed to intersect. Since diagonals are all parallel this means that no diagonal is allowed to cross a rectangle-segment. Assume that the diagonal "step-line" of a block a intersects with the rectangle-segment of a block b. If we break the course of the diagonal of a and let it continue along the rectangle-segment of block b and further along the diagonal of b we would not violate the gridding rules since no-segment would cross. This is illustrated on figure 4.6(b).

We are now able to reason on this method. If a diagonal of a rectangle a intersects with a rectangle-segment of a rectangle b then the remainder of the positive upper-right "step line" of a can run along the "step-line" of b. If the diagonal of a hits the rectangle-segment on the upper-left side of the rectangle-segment, then a must precede b in $\mathbf{A}$. On the other hand if the diagonal of a hits the rectangle-segment on the lower right side, then a must succeed b in $\mathbf{A}$. We will clarify this in a moment.

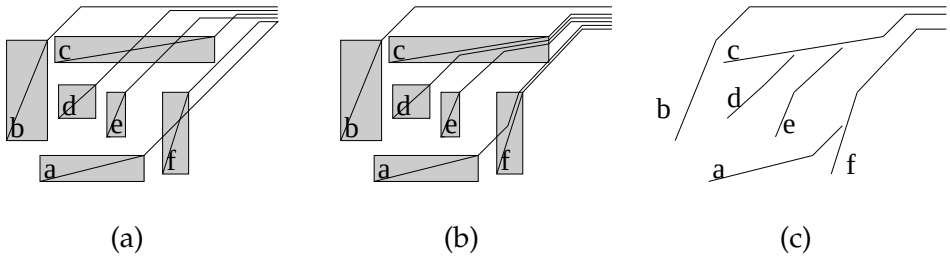


Figure 4.6: *Illustration of the diagonals and rectangle-segments. (a) The diagonals are now drawn from the upper right corner of the modules and segments are added between lower-left and upper right of the rectangles these are called "rectangle-segments". (b) No diagonals are allowed to intersect a rectangle-segment – This is the correct layout of the diagonal "step lines". (c) Diagonal step lines are cut and merged at intersections.*

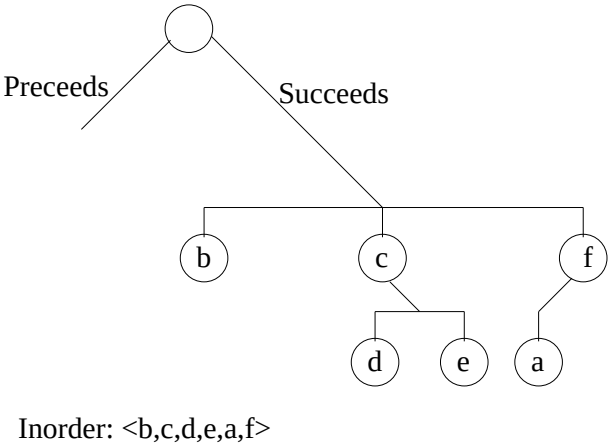


Figure 4.7: *Tree like structure to describe the* **A***-sequence of the modules from figure 4.6. A "node" in the tree represent a module and has two children. The left child is an ordered list of modules preceding it. The right child is an ordered list of modules succeeding it.. The top node is an artificial node and every module succeeds it. An "in-order" run gives the* **A***-sequence.*

Based on this discussion we can thus break the positive ''step-line" of a once it hits a rectangle-segment. This is illustrated on figure 4.6(c).

In figure 4.6(c) the diagonals of d and e hits the rectangle-segment of c from lower right side so these to modules succeed c in that sequence. Also the diagonal of a hits the rectangle-segment of f from top left side and so a should precede f. This can be represented by a tree-like structure as shown on figure 4.7. An extra artificial rectangle-segment is added at the top corresponding to the root of the tree.

Based on this discussion we can present the details of our sweep-line algorithm.

First we rotate the coordinate system so that the diagonals become parallel with the y-axis (see figure 4.8). An orthogonal coordinate system with the diagonals as one axis is given by the vectors $(1, -\alpha)^t$ and $(\alpha, 1)^t$. It is therefore easy to verify that the

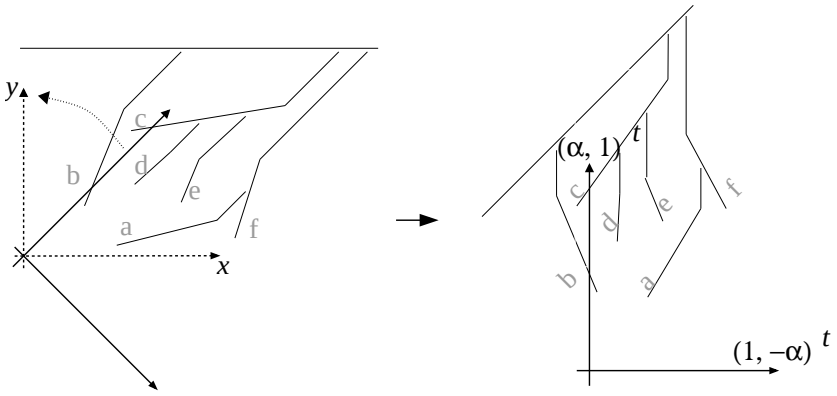


Figure 4.8: *Rotation of the coordinate system such that the diagonals become parallel to the y-axis. The top line-segment is the top of the packing area corresponding to the artificial module.*

coordinate system transformation induced by the matrix

$$\begin{pmatrix} 1 & -\alpha \\ \alpha & 1 \end{pmatrix}, \tag{4.2}$$

rotates diagonals into the y-axis. Note that the transformation is not distance preserving. Now the algorithm proceeds as a common sweep-line algorithm.

A sweep-line algorithm moves from left to right only stopping at specific x-coordinates which are called *breakpoints* (see e.g. [13] for an introduction to sweep-line algorithms). In our case the breakpoints are at the transformed lower-left and upper-right corners of rectangles. The breakpoints serve two purposes. Firstly whichever of the two breakpoints comes first activates the rectangle-segment of the corresponding rectangle. Activation inserts the rectangle-segment into a data structure containing rectangle-segments ordered by their transformed y-coordinate. The last of the two breakpoints removes the segment from the data structure. Further the upper-right breakpoint for a rectangle a checks which rectangle-segment is above it in the transformed coordinate system. Let b be the rectangle of the rectangle-segment which is above the upper right corner of a. If the rectangle-segment above is descending (has negative slope) then a is above b. If it is ascending (positive slope) then a is below b. In both cases a is inserted into the previously described tree-structure in either preceding or succeeding list. Figure 4.9 illustrates this.

Finally when all breakpoints have been visited the nodes in the tree-like structure are visited in an in-order search. The in-order search starts from the top of the tree-like structure and visits each node recursively by first visiting predecessor nodes, then the node itself and finally successor nodes.

A more precise description of the algorithm is given in algorithm 4.2.

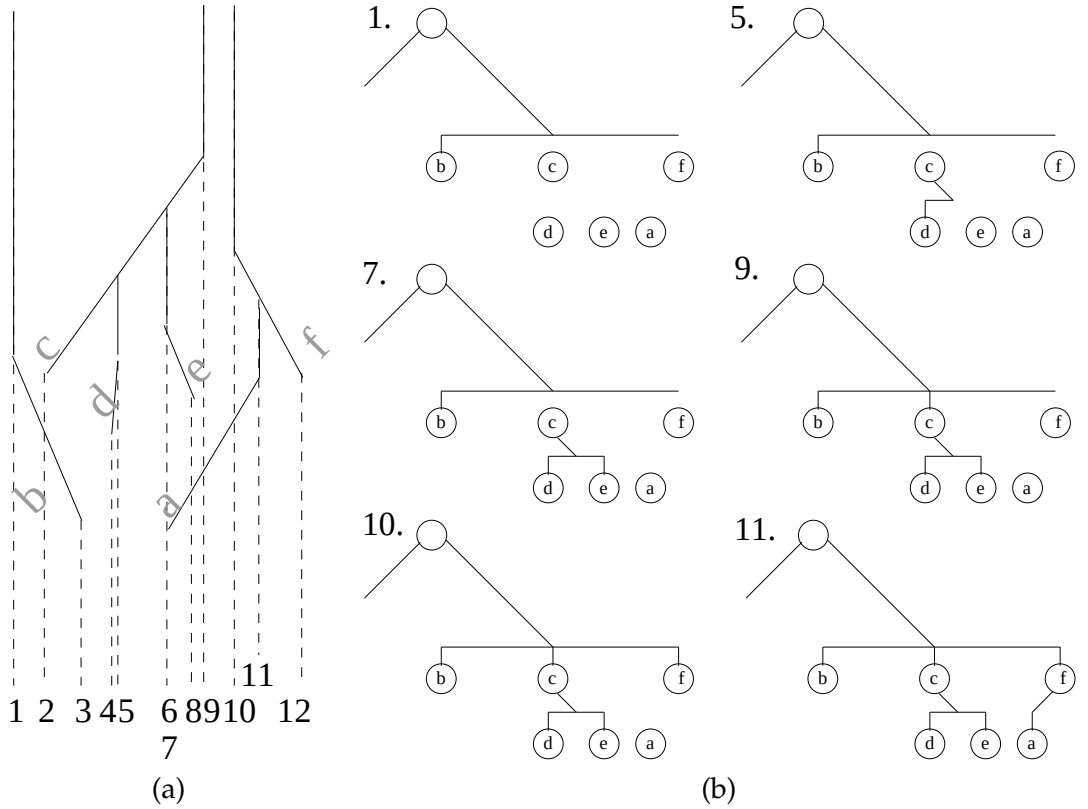


Figure 4.9: *The sequence-pair sweep-line algorithm at work. (a) Breakpoints in the transformed coordinate system. (b) The evolving tree-like structure. The numbers in upper right represent the breakpoint of the occurrence. Note that the structure only changes at upper-right breakpoints. The lines of the tree in (b) show how rectangles get connected at each breakpoint.*

Asymptotic Running Time The asymptotic running time is dominated by sorting the breakpoints and maintaining the rectangle-segment data structure. The rectangle-segments cannot cross since this would imply overlap in the placement and we assumed that the placement contained no overlap. With this in mind both sorting and maintaining the rectangle-segment structure can be achieved in $O(n \log n)$ time if a balanced tree structure e.g. red-and-black tree (see [13]) or similar is used for the active rectangle-segments.

This algorithm creates the $\mathbf{A}$-sequence of the modules. By mirroring the modules in the y-axis one can use the same algorithm to encode the $\mathbf{B}$-sequence.

4.3.3 Overlapping Placements

Although the previous algorithm does, to some extend, work for placements with overlap have taken no special care to handle it. Therefore we we will present an algorithm in this section that deals with overlap in a structured fashion.

Algorithm 4.2: Algorithm for sequence-pair encoding

Input(*A placement* $P = (x, y)$ *of a set of placed rectangles* R *with coordinates* $(x(r), x(y))^t$ $r \in R$*, and a value* $\alpha \in \mathbb{R}$) ;
Generate breakpoints at the lower-left and upper-right transformed corners of the rectangles in R ;
Sort the breakpoints by increasing transformed x-coordinate;
foreach *Breakpoint* **do**
  if *Breakpoint is start of rectangle-segment* **then**
    Insert the rectangle-segment into segment data structure.
  if *Breakpoint is end of rectangle-segment* **then**
    Remove the rectangle-segment from segment data structure.
  if *Breakpoint is upper-right of rectangle* r **then**
    Find segment s above ;
    Let t be the rectangle of s;
    if s *has negative slope in transformed coordinate system* **then**
      Insert r as successor of t
    else
      Insert r as predecessor of t
Do in-order search of tree-like structure;
return *The sequence of modules visited during in-order search.*

There are at least two approaches; either we extend algorithm 4.2 to handle overlap or we create a completely new algorithm which runs prior to algorithm 4.2 and removes any overlap.

We have chosen the second strategy. There are three reasons for this.

- A separate algorithm keeps algorithm 4.2 simple,
- a separate algorithm to handle overlap may work regardless of the abstract data-structure used (e.g. sequence-pair, B*-tree, O-tree etc.), and is therefore a more general approach,
- and preliminary attempts at extending 4.2 proved that consistent overlap handling was difficult.

If we are not allowed to move the modules there is only one way to remove overlap; to reduce their size. This is our strategy. Before we describe the method we will introduce a concept which is commonly used for the nesting problem (polygon packing). It should be noted that we have adapted the concept slightly.

Definition 4.7. Intersection depth Let rectangles a and b overlap. Then we say that the intersection depth along a vector v is the distance a has to move along v so that a and b no longer overlap.

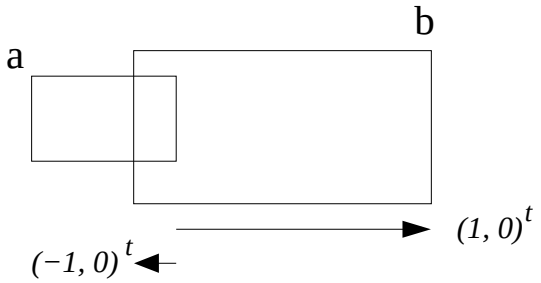


Figure 4.10: *Intersection depths between a and b along the vectors $(1,0)^t$ and $(-1,0)^t$. I.e. moving a right or left.*

In nesting problems this has led to the no-fit polygon which can be used to quickly determine the intersection depth. Without dwelling upon it further the no-fit polygon of two rectangles with dimension $(a_w, a_h)^t$ and $(b_w, b_h)^t$ is simply a rectangle with dimensions $(a_w + b_w, a_h + b_h)^t$. Instead of using the no-fit polygon we will make a few observations with respect to intersection depth along the x- and y-axis.

Definition 4.8. Minimum intersection depth Let a and b be overlapping rectangles in a placement $P = (x, y)$. The minimum intersection depth along the x-axis is the smallest of the intersection depths along the vectors $(1,0)^t$ and $(-1,0)^t$. (I.e. moving left or right). The minimum intersection depth along the y-axis is defined similarly.

Lemma 4.1. *If a and b are rectangles in a placement $P = (x, y)$ with dimension $(a_w, a_h)^t$ and $(b_w, b_h)^t$ and lower-left coordinates $(x(a), y(a))^t$ and $(x(b), y(b))^t$ and the center of a is left of the center of b, i.e. $x(a) + \frac{a_w}{2} \leq x(b) + \frac{b_w}{2}$ then the minimum intersection depth along the x-axis is along the vector $(-1,0)^t$.*

Proof. There are only two possibilities. The intersection depth along the vector $(1,0)^t$ is $x(b) + b_w - x(a)$ and the intersection depth along $(-1,0)^t$ is $x(a) + a_w - x(b)$ (see figure 4.10). We deduce:

$$\begin{aligned} x(a) + \frac{a_w}{2} &\leq x(b) + \frac{b_w}{2} \\ 2x(a) + a_w &\leq 2x(b) + b_w \\ x(a) + a_w - x(b) &\leq x(b) + b_w - x(a) \end{aligned} \tag{4.3}$$

□

This lemma is the base of our algorithm. From it we can deduce that if the center a is left of the center of b then moving a to the *left* until a and b no longer overlap is less movement than moving a right until a and b no longer overlap. Because of the "antisymmetry" of intersection depth we know that the opposite is also true.

Running time of our algorithm is important. Since the other algorithms (non-overlapping placement to sequence-pair and sequence-pair to placement) have running times less than or equal to $O(n \log n)$ we wish to hit the same running time for this algorithm which limits our possibilities.

Our algorithm consists of two almost identical sweep-line algorithms. One works from left to right. The other from bottom to top. We begin by describing the left to right algorithm.

First we introduce a cut-method between two overlapping rectangles. The method either cuts the rectangles by a vertical cut-line or by a horizontal cut-line. We call the vertical cuts x-cuts and horizontal cuts y-cuts. Let us consider the x-cuts since the y-cuts are rotated but otherwise exactly the same. For a rectangle a in a placement $P = (x, y)$ let $a^x = [x(a), x(a) + a_w]$ and $a^y = [y(a), y(a) + a_h]$.

We consider two rectangles a and b. Assume also that the center of a is left of the center of b. According to the previous discussion it makes sense that a should be left of b after the cut. When considering the intervals a^x and b^x there are now three possibilities.

- $a^x \subseteq b^x$ In this case we cut halfway between the centers but no more right than $x(a) + a_w$.
- $b^x \subseteq a^x$ Here we cut halfway between the centers but no more left than $x(b)$.
- $x(b) \geq x(a) \wedge x(b) + b_w > x(a) + a_w$ We cut halfway between left side of b and right side of a.

Note that the remaining possibility $x(a) \geq x(b) \wedge x(a) + a_w > x(b) + b_w$ would imply the center of a is right of the center of b. The three possibilities are depicted on figure 4.11 and allow us to define a function $x\mathrm{cut}(a, b)$ which returns the x-coordinate of the vertical cut-line between a and b. Note that $x\mathrm{cut}(a, b)$ assumes that a should be left of b after cutting and the function is not symmetric. We also introduce a function $y\mathrm{cut}(a, b)$ which is equivalent to $x\mathrm{cut}(a, b)$ but considers y-direction instead of x-direction.

For two rectangles a and b we also wish to determine whether an x- or a y-cut is best. We considered three ways to choose between x- and y-cuts:

1. Choose the direction that maximizes area after cut.
2. Choose the direction with smallest intersection depth.
3. Choose the direction with relative smallest intersection depth. I.e. $\frac{x_{\mathrm{depth}}}{x_{\mathrm{maxdepth}}}$ or $\frac{y_{\mathrm{depth}}}{y_{\mathrm{maxdepth}}}$

In all cases we assume that centers specify which rectangle should be below or left as discussed in lemma 4.1. Examples of the three possibilities are shown of figure

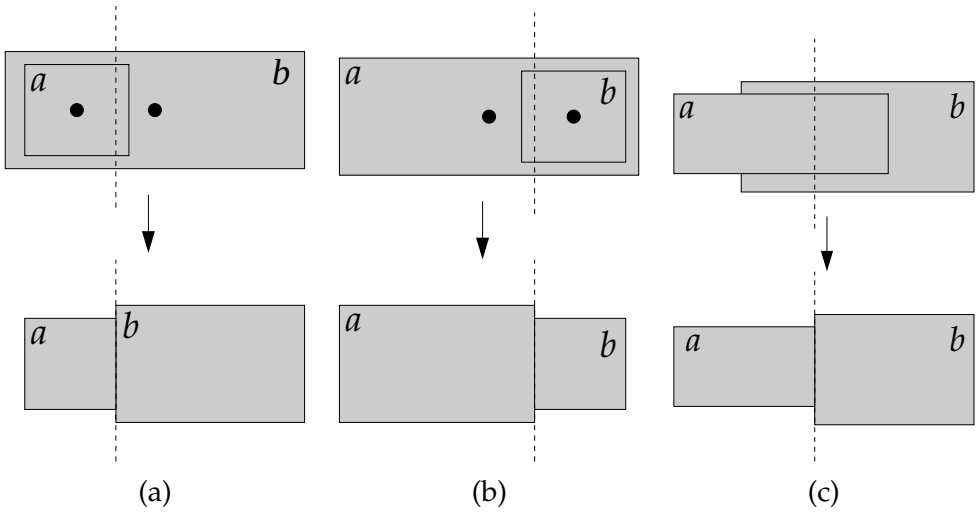


Figure 4.11: *The three possible x-cuts (see the text). (a) $a^x \subseteq b^x$. (b) $b^x \subseteq a^x$, (c) $x(b) \geq x(a) \wedge x(b) + b_w > x(a) + a_w$*

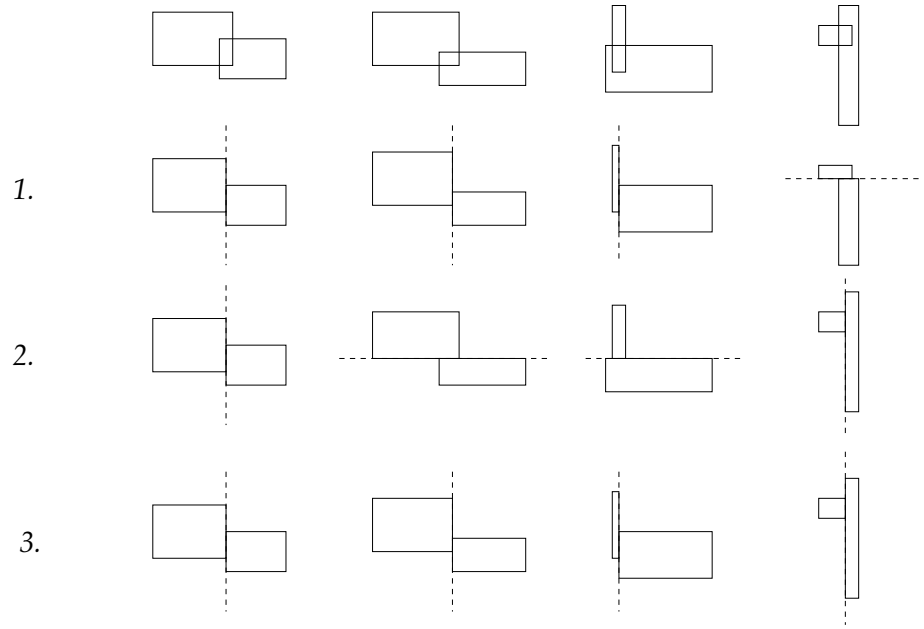


Figure 4.12: *Examples of three different ways to determine whether to do an x-cut or a y-cut. Top row is the original overlapping packing and the other tree rows are various solutions. 1. is based on area. 2. based on direction with smallest minimum intersection depth. 3. smallest relative minimum intersection depth.*

4.12. Based on investigations we have decided to use the smallest relative minimum intersection depth method. Let the function $xy\text{cut}(a, b)$ determine whether to do an x-cut or a y-cut of the overlapping rectangles a and b.

We are now ready to explain our cut-algorithm which is also a sweep-line algorithm. We visit the rectangles of the placement from left to right sorted by center-coordinates. At any given time during the algorithm an interval data structure A contains a set of intervals sorted by y-coordinate. Each interval correspond to a rectangle and we call the rectangles which have intervals in A *active rectangles*.

When we visit a rectangle b we search for its center y-coordinate in the interval structure. First we must determine the cut-line between b and every overlapping rectangle.

To do this we move from the center to the bottom of b visiting intervals from A. Each interval corresponds to an active rectangle a which *overlap vertically* with b ($a^y \cap b^y \neq \emptyset$). For every such rectangle a encountered during this search which also overlap with b we determine – by $xy\text{cut}(a, b)$ – whether an x- or a y-cut is best. If an x-cut is best we calculate the cut-line by the $x\text{cut}(a, b)$ function. We store the rightmost cut-line. When testing with the $xy\text{cut}$-function we always pretend that b has been cut by the rightmost cut-line encountered so far. This downwards search pauses when $xy\text{cut}$ reveals that a y-cut is best or we have reached the lowest interval which overlap vertically with b.

We now move from center to top and calculate cut-lines in the same manner as from center to bottom. Once again we investigate for each rectangle of the interval structure whether an x- or a y-cut is best.

When we have finished the center to top search we resume the center to bottom search if it was paused by suggesting a y-cut. The reason for this is that the center to top search may have moved the rightmost cut-line further right and could therefore influence the decision made by $xy\text{cut}$. Similarly we may resume the center to top search since the second center to bottom search may also have moved the cut-line. This continues until the searches in neither direction visits new rectangles.

When the two cut-line determination searches completes we do a second search which actually cuts all rectangles according to the rightmost cut-line. However we do no y-cuts yet. All intervals $I = [a_b, a_t]$ in A with $I \subseteq b^y$, i.e. covered by b are removed from A and finally an interval I_b for b is inserted into A.

Intervals overlapping with I_b are cut so the interval structure never contains overlapping intervals. After deletion there can be a maximum of two intervals from the interval data-structure which overlap with I_b. If there is only one interval I it may contain I_b completely. In this case we break I in two pieces. A lower and an upper piece (see figure 4.13). Otherwise we simply break intervals so they no longer overlap.

The method is described in algorithm 4.3.

Note that if we decide to do a y-cut, b from the outer loop may overlap with either the rectangle of *top* or *bottom*. In this case we should be careful when updating A.

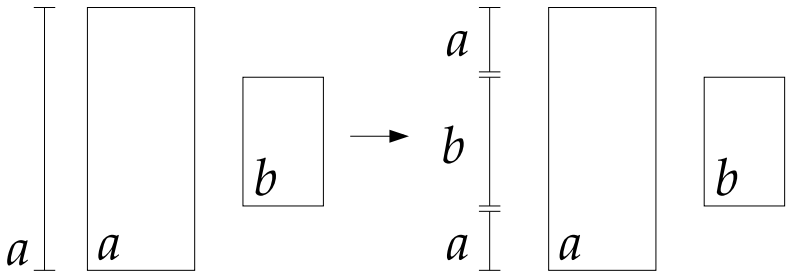


Figure 4.13: *Insertion of rectangle b in interval structure. The interval for a completely contains the vertical extend of b. Therefore it is split in two intervals.*

Algorithm 4.3: Algorithm for removing overlap by x-cutting

Input(*A set of rectangles R in a placement $P = (x, y)$*) ;
A := [Data-structure containing intervals for active rectangles ordered from top to bottom];
L = [R sorted by center x-coordinate] ;
foreach *rectangle* $b \in L$ **do**
  Let $y = y(b) + \frac{b_h}{2}$;
  Let $x' = x(b)$;
  Determine highest interval $I_c = [y_c^b, y_c^t] \in A$ such that $y_c^b \leq y$;
  Let $bottom = I_c$;
  Let $top = I_c$;
  repeat
    repeat
      Let r be rectangle for $bottom$. **if** r *overlaps with* b **then**
        if $xy\text{cut}(r, b)$ *determines x-cut best* **then**
          Let $x' = max(x', x\text{cut}(r, b))$;
        else
          Break
      $bottom$ = [Interval below $bottom$]
    until $bottom_y \leq y(b)$;
    [Do as above but go upwards updating *top*]
  until *top or bottom not changed*;
  [Cut b according to cut-line at x'] ;
  [Cut visited rectangles in A between $bottom$ and top according to cut-line] ;
  [Remove intervals in A between $bottom$ and top covered by b] ;
  [Insert b into A] ;
  [Update bottom and top intervals] appropriately

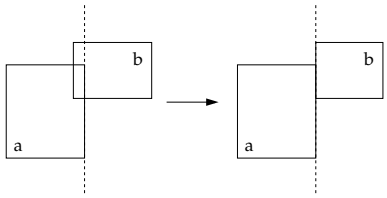


Figure 4.14: *During the y-run we may wish to do an x-cut instead of a y-cut. Rather than postponing this cut, we do a simple x-cut using the lower (with respect to center) rectangle's left or right edge.*

When updating the intervals top and $bottom$ we let the rectangle with rightmost side dominate. So assume that b overlaps with e.g. the rectangle of top then we update the intervals for b and top such that the interval $I = [y_i^b, y_i^t]$for the rectangle with rightmost side is maintained while the other interval is broken so it no longer overlaps with I.

To handle the y-cuts which were all postponed during the x-cutting algorithm, we do a similar run in the y-direction where rectangles are sorted by y-coordinate of center and with y-cuts. The only difference in this case is that should $xy\text{cut}$ determine that an x-cut is best we do a vertical cut immediately rather than postpone. Assume that we visit a rectangle b which is compared with a rectangle a then if a and b overlap and $xy\text{cut}$ determines that we should do an x-cut then if a is left of b we cut at the right edge of a and if a is right of b we cut at the left edge of b (see figure 4.14).

Proceeding the y-cutting algorithm no overlap remains. This statement is not trivial. Let us consider how the cuts work during the y-cutting algorithm. In this case a rectangle b is inserted and compared against rectangles $a \in A$. Let us start by considering y-cuts. Proceeding the inner loops b cannot overlap with any rectangle visited before b. The primary reason for this is that all three possibilities for a y-cut cuts above a's center disallowing b to reach below a. Thus it is impossible for b to touch any rectangle covered by a. So after the cut b cannot touch any rectangle which has been removed from the active rectangles set A. On the other hand the cut line is chosen such that b cannot overlap with any rectangle from A either since we also dealt with x-cuts immediately.

The algorithm is connected to lemma 4.1. If two rectangles overlap the algorithm maintains their horizontal or vertical order with respect to their centers. If the rectangles were re-expanded to full size and this order was maintained it would mimic a move of the rectangles in the direction of the minimum intersection depth of either horizontal or vertical direction.

Running time Each of the two sweep-line algorithms have running time $O(n \log n)$ where n is the number of rectangles as promised. The active rectangles interval data-structure, A, can be implemented with a balanced search tree – e.g. a red and black tree – indexed by lower coordinates of the intervals. The search and insert operations can be done in

$O(\log n)$ time. Insertion of an interval for a rectangle can only give rise to a maximum of two new intervals in the interval data-structure if another interval is split. Therefore no more than $2n$ intervals can be inserted. Apart from no more than two intervals in each iteration $bottom$ and top of the outer loop all intervals in A are visited only once since the removal step removes all intervals strictly between top and bottom. The total interval visits must therefore be $O(n)$ so the amortized cost of each insertion is $O(\log n)$ time and the algorithm has running time $O(n \log n)$.

An example of insertion of one rectangle is shown in figure 4.15.

4.4 From Sequence-Pair to Placement

Since the introduction of the sequence-pair in [63] several algorithms for converting a sequence-pair to an actual placement have been proposed. An incomplete survey of the placement algorithms were given in section 3.4.1. In this section we will describe a new extended version of a placement algorithm proposed by Pisinger [72]. However to help the reader understand the algorithm we will explain the three previous most successful algorithms first.

4.4.1 Previous Sequence-Pair Placement Methods

Constraint Graphs

The original method used by Murata et al. was based on longest paths in a graph. Theorem 4.1 describes rectangle relations. For each rectangle a it can be examined which rectangles are to the right and above a. A vertical and a horizontal constraint graph can now be constructed from the rectangle relations. In this context a constraint graph is a weighted, acyclic, directed, graph.

The horizontal constraint graph can be constructed from the sequence-pair $\langle \mathbf{A}, \mathbf{B} \rangle$ as follows:

1. For each rectangle a add a corresponding node.
2. Add two extra nodes W (west) and E (east).
3. Add a directed edge between nodes a and b if rectangle a is left of b as determined by the sequence-pair $\langle \mathbf{A}, \mathbf{B} \rangle$ using theorem 4.1. Set its weight to the width of a.
4. Add a directed edge between W and every node with weight 0.
5. Add a directed edge between every node and E with weight 0.

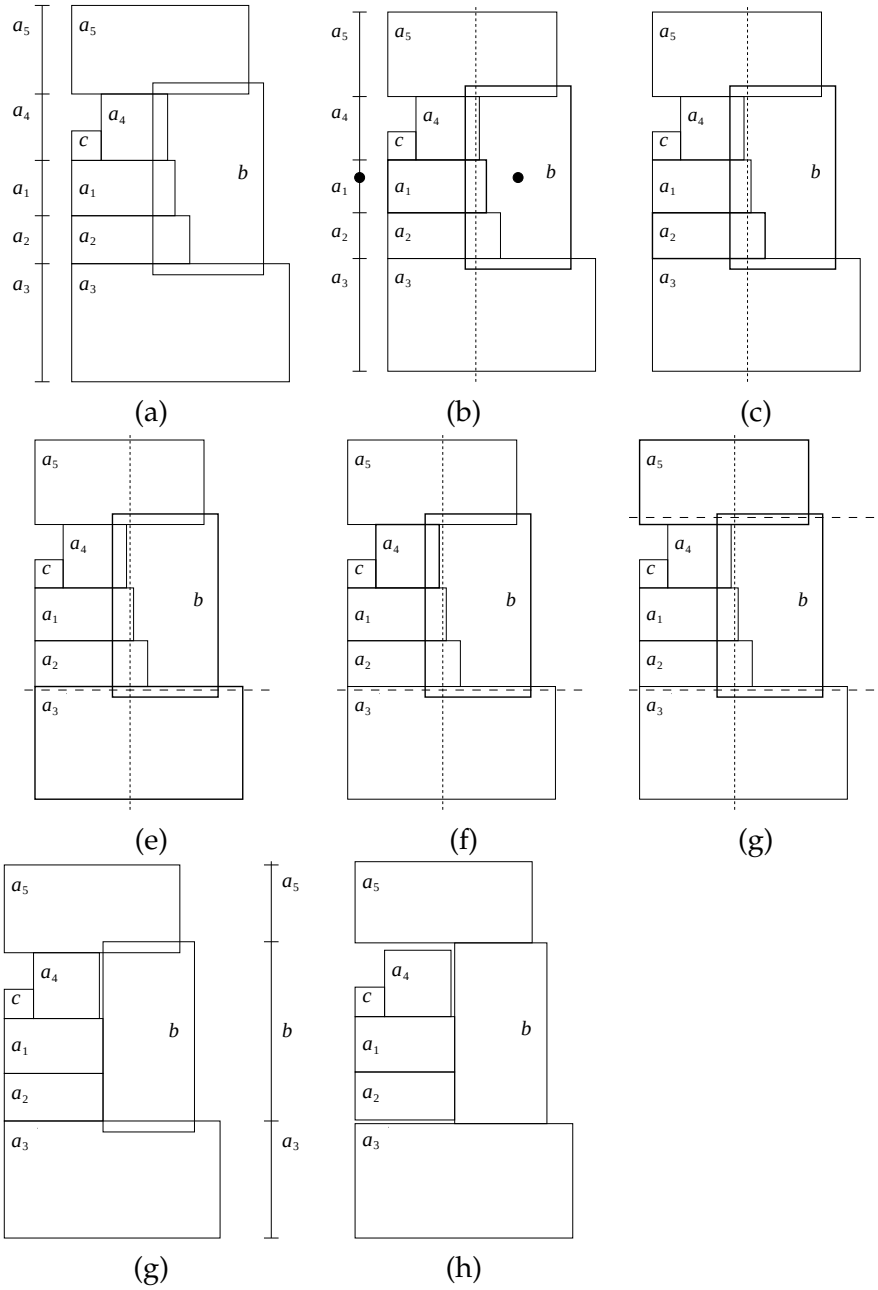
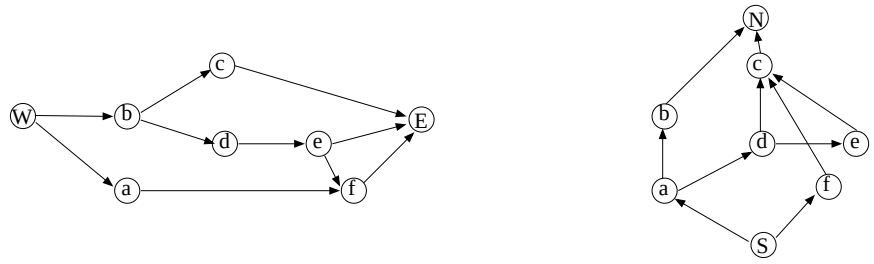


Figure 4.15: *Visiting a rectangle b in algorithm 4.3. (a) The setup. Rectangle b is to be visited. Rectangles $a_1, \ldots, a_5$ are active and the interval data structure with active rectangles is shown on the left. c is a non-active rectangle. (b) First the starting interval is for the rectangle a_1 since a_1's interval overlaps with b's center. The cut-line is calculated (dashed line). (c) b is checked with a_2 and the cut-line is moved slightly to the right. (d) b is checked with a_3 and it is determined that a y-cut is best. The downward search is suspended. (e) b is checked with a_4. No change. (f) b is checked with a_5 and a y-cut chosen. This ends the search for the cut-line. (g) The rectangles are cut against the cut-line and the interval data structure is updated. Notice that a_3 has higher priority when updating the segment containing b and a_3 because a_3's right edge is further to the right than b's. (h) The placement after the succeeding y-cut algorithm.*



(a) (b)

Figure 4.16: *Constraint graphs of the sequence-pair from figure 4.1. Weights and transcendent edges are omitted. (a) is the horizontal constraint graph. (b) is the vertical constraint graph.*

The vertical constraint graph is created in a similar fashion but with W and E replaced with S (south) and N (north) and edges are added on the below relation instead of the left relation with weight equal to the height of rectangles instead of their widths. Figure 4.16 shows the constraint graphs of the sequence-pair of the placement of figure 4.1.

We can now determine the position of a rectangle a. Let s be the length of the longest path from W to a in the horizontal constraint graph and t be the length of the longest path from S to a in the vertical constraint graph. Then the lower left coordinate of a is (s, t). This is justified by the fact that a must be placed right and above all the rectangles left of and below it, which is exactly the rectangles b for which there is a directed edge (b, a) in the constraint graphs.

Determining the longest path from E and S to every other node in respectively the horizontal and vertical graph can be done in time $O(n^2)$ since there are $O(n^2)$ edges. This is the running time of the original sequence-pair placement algorithm. The required placement area size of a sequence-pair is determined by the longest path from W to E and from S to N.

Weighted Longest Common Subsequence

Tang and Wong [81] used the observation from theorem 4.1 quite differently than Murata et al. They observed that given a rectangle a the set of rectangles to the left of a are those rectangles which precedes a in both sequence $\mathbf{A}$ and $\mathbf{B}$. If each rectangle is given weight equal to their width in the two sequences, the horizontal position of a can be determined as the weighted longest common subsequence of the rectangles preceding a in $\mathbf{A}$ and $\mathbf{B}$. Similarly, using the second part of theorem 4.1, we get that the vertical position of a can be determined from the longest common subsequence of the rectangles preceding a in $\mathbf{A}$ and preceding a in the reverse of $\mathbf{B}$.

An algorithm to determine the weighted longest common subsequence of $\mathbf{A}$ and $\mathbf{B}$ is

presented in [81] which runs in time $O(n \log n)$. Tang and Wong [82] later improved this to $O(n \log \log n)$ using a priority queue data structure.

Confined and fixed rectangles In [82] Tang and Wong were also able to confine rectangles to regions and specific positions. Confining a rectangle a to the region between (x_0, y_0) and (x_1, y_1) was done simply by introducing a dummy rectangle between W and a with width x, a dummy rectangle between a and E with width equal to $w - x_1$ (w is the width of the placement area) etc. This method of confinement however comes at a price. It is possible that the sequence-pair representation does not allow a rectangle a to be placed in its confined region; e.g. if there are too many rectangles to the left of a or the dummy rectangle between a and E is too far right. Tang and Wong solves this problem by simply disallowing such placements which ruins the P-admissibility (see section 4.2) of the resulting solution space.

Semi-Normalized Placement

Pisinger [72] presented a method with equal running time of Tang. Here a so called envelope is used to guide the placement. Rather than determining the position of each rectangle independently, Pisinger places the rectangles one at time and determines the position of new rectangles from previously placed rectangles.

When placing a rectangle the legal positions are "open" corners between already placed rectangles. The positions corresponds to placing a rectangle as far down and left as possible but without allowing rectangles lower of more left than right and upper edges of previously placed rectangles.

We will explain the algorithm in detail since it will be the foundation of our new slightly more complicated placement method. We begin with the following definitions.

Definition 4.9. Rectangle-shade A rectangle a with width a_w and height a_h is *shaded* by a rectangle b in a packing $P = (x, y)$ if and only if

$$\begin{aligned} x(b) + b_w &\geq x(a) + a_w \\ \text{and} \quad y(b) + b_h &\geq y(a) + a_h \end{aligned} \tag{4.4}$$

In addition we say that a rectangle a is shaded if there exists a rectangle b such that a is shaded by b. See figure 4.17(a).

Definition 4.10. Corner between rectangles The corner of the ordered pair of rectangles (a, b) in a packing $P = (x, y)$ is the point $(x(a) + a_w, y(b) + b_h)$. See figure 4.17(b)

The algorithm constructs a new packing by iteratively adding a rectangle to an existing packing. At any time during the constructive placement algorithm a data structure

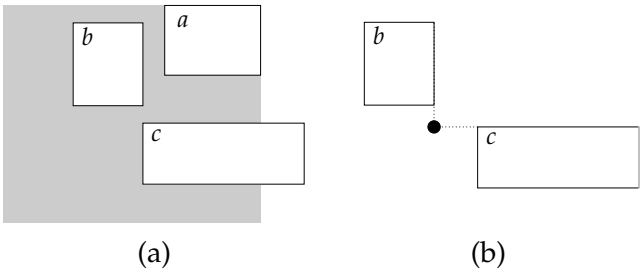


Figure 4.17: *(a) Shade definition. In this packing a shades b but not c. All rectangles contained completely within the grey area would be shaded by a. (b) corner definition the dark circle is the corner between rectangles a and b.*

E, called the envelope, contains the rectangles that are not shaded with respect to the current packing. The rectangles in E are ordered and the corners of succeeding rectangles of E are candidate positions for the lower left corners of rectangles to be added. So if e.g. $E =< e_1, e_2, e_3, \ldots, e_k >$ all corners of rectangle pairs (e_i, e_{i+1}) are candidate positions. Initially two dummy rectangles s and t are added to E. s is infinitely high and left of all rectangles and t is infinitely wide and below all rectangles. So the initial envelope is $E =< s, t >$.

To determine the right corner to place a rectangle a we find a rectangle pair (e_i, e_{i+1}) from E such that e_i is left of a and e_{i+1} is below a according to the sequence-pair. For now assume such a pair exists.

Now we position a such that its lower left corner is exactly the rectangle corner of (e_i, e_{i+1}). We then update E by inserting a between e_i and e_{i+1} and remove all rectangles shaded by a. The algorithm is sketched in algorithm 4.4.

The algorithm accepts as input a sequence pair $\langle \mathbf{A}, \mathbf{B} \rangle$ of rectangles. The rectangles are placed in order according to the $\mathbf{B}$-sequence and positioned according to the $\mathbf{A}$-sequence .

Lets us briefly return to our assumptions regarding the uniqueness of (e_i, e_{i+1}). Whenever we place a rectangle a let us pick the e_i of E which has the highest index less than the index of a in the $\mathbf{A}$-sequence. Initially $E =< s, t >$ and for any rectangle we would have $e_i = s$. Because we always place a rectangle according to this rule the elements of E will be sorted according to their indices in the $\mathbf{A}$-sequence. Since every element has unique index in the $\mathbf{A}$-sequence we are guaranteed uniqueness.

This procedure makes sense. When we place a rectangle a any rectangle in E has lower index in the $\mathbf{B}$-sequence. If a rectangle $e_i \in E$ has lower index than a in the $\mathbf{A}$-sequence we know from 4.1 that a should be placed to the right of e_i. Similarly if e_{i+1} has higher index than a in the $\mathbf{A}$-sequence we know that a should be placed above e_{i+1}.

Of course the dominant part of the running time of the algorithm lies in determining the predecessor e_i of a rectangle in E and removing newly shaded rectangles. We will

Algorithm 4.4: Semi-normalized sequence-pair placement

Input(*Sequence-pair:* $A =< a_1, \ldots, a_n >, B =< b_1, \ldots, b_n >$);
E = < s,t >;
for k = *1* ***to*** n **do**
  $h = b_k$;
  Determine $e_i \in E$ with highest index less than h's index in A ;
  $p = e_i$;
  $q = e_{i+1}$;
  $x(h) = x(p) + p_w$;
  $y(h) = y(q) + q_h$;
  [insert h in E between p and q]. ;
  DeleteShadedRectangles(E, h) ;

now explain these parts in more detail.

Since elements of E are sorted with respect to their index in the **A**-sequence E can implemented as a search-tree data-structure. Also note that the elements of E have **A**-indices from $\{1, \ldots, n\}$ Pisinger suggested the priority-queue structure by Van Emde-Boas et al. [84]. When the elements of the priority-queue have values from the set $\{1, \ldots, n\}$ the operations **insert** and **delete** can be done in time $O(\log \log n)$. In other words the Van Emde-Boas structure enables determination of predecessor and insertion of new rectangle to be done in time $O(\log \log n)$.

The last step of each iteration – removing shaded rectangles – can also be implemented quite efficiently. Whenever a rectangle h is inserted the successors of h in E are traversed until a non-shaded rectangle is reached. This is sufficient because once a a non-shaded successor rectangle i is reached no further successors from E can be shaded by h since then they would also be shaded by i and have been removed when i was inserted (see figure 4.18). Similarly the predecessors are traversed until a non-shaded rectangle is reached. This is also sufficient.

Except from the last successor and last predecessor all rectangles visited during this traversal are removed from E. A rectangle can only be removed from E once, so the total number of visits during the placement must be less than $2n + n = O(n)$. In other words the removal takes amortized constant time for each rectangle.

An example of the semi-normalized placement of a sequence-pair is shown in figure 4.19.

The semi-normalized placement does not in strict sense place according to the sequence-pair. A simple example for which this is not the case is shown on figure 4.20. Notice how the semi-normalized placement compacts the sequence-pair by moving rectangles down and left. Although the semi-normalized placement does not obey the sequence-pair it is P-admissible and it does seem to produce more compact placements.

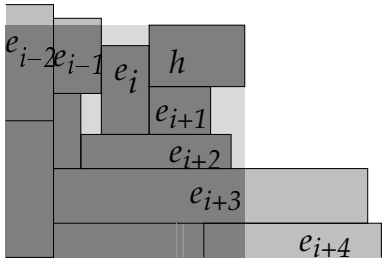


Figure 4.18: *Removing the shaded rectangles during semi-normalized placement. the rectangle h is being inserted between e_i and e_{i+1}. The removal algorithm picks predecessors and successors of h until it reaches a rectangle which is not completely shaded. In this case the rectangles e_i and e_{i+1}, e_{i+2} are removed from the envelope.*

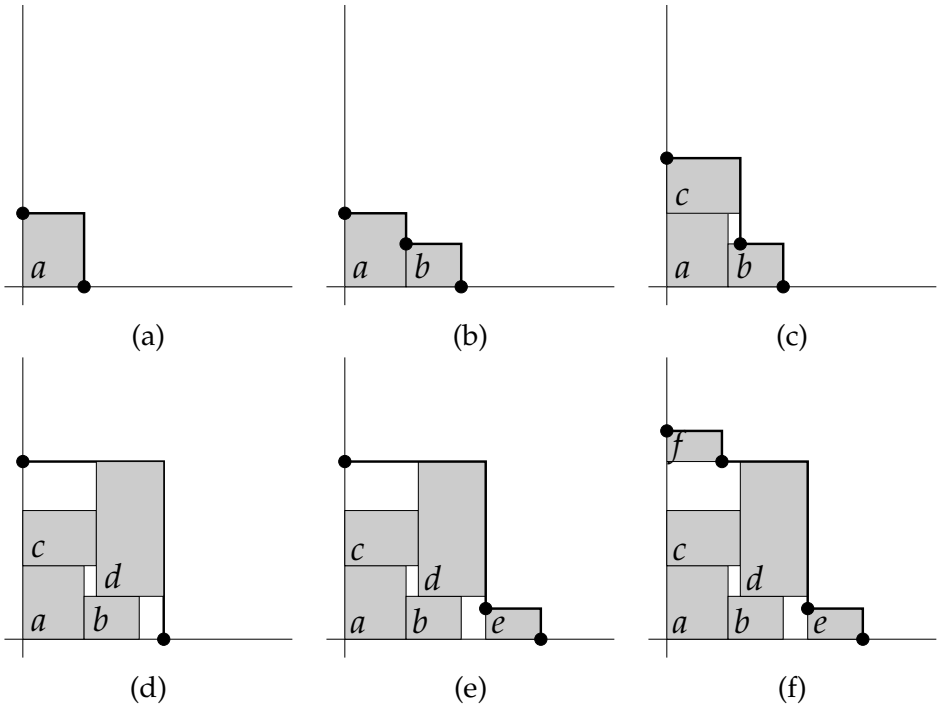


Figure 4.19: *The semi-normalized placement in action. Six rectangles are placed according to sequence-pair $\langle < c, f, d, a, b, e >, < a, b, c, d, e, f > \rangle$ Each rectangle can only be placed at the black circles. The thick dark line is the "envelope". No rectangle can be placed below or left of the envelope, and the corner points of the envelope are legal positions. Although the semi-normalized placement is more compact than a regular sequence-pair placement rectangles still "float" in midair ((e) and (f)).*

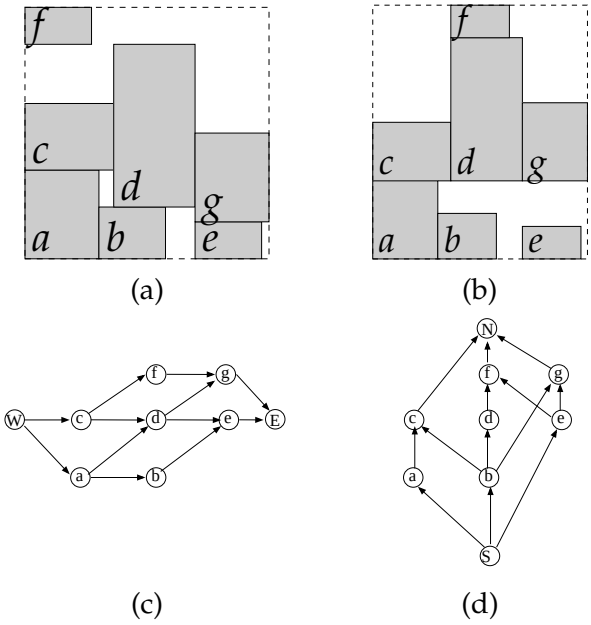


Figure 4.20: *Comparison of semi-normalized placement (a) and strict sequence-pair placement (b) of the sequence-pair* $\langle < c, f, d, g, a, b, e >, < a, b, c, d, e, f, g > \rangle$*. Notice how rectangle* d*,* g *and* f *floats in the strict sequence-pair placement. (c) and (d) are constraint graphs for the placement. In other words the semi-normalized placement violates some of the constraints of the sequence-pair to produce a more compact placement.*

4.4.2 Extended Semi-Normalized Placement

In this section we present a new placement method which can produce placements that are slightly more compact than the semi-normalized method while sharing the asymptotic running time.

First we define rectangle-visibility.

Definition 4.11. Visibility of a rectangle. A rectangle r in a packing $P = (\overline{x}, \overline{y})$ is said to visible from above if there exists an infinite vertical line segment

$$l : \begin{pmatrix} x \\ y \end{pmatrix} = t \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} x' \\ r_y + h \end{pmatrix}, \quad t \in \mathbb{R}_+ \tag{4.5}$$

with $x' \in [\overline{x}(r), \overline{x}(r) + r_w]$, and which does not intersect any rectangle (see figure 4.21). We define visibility from left, right and below similarly.

Now the idea is to introduce two additional sets of extreme rectangles. Let E_h be the rectangles of the current placement which are visible from above and E_v be the rectangles visible from the right. This will enable us to "push" the rectangles either downwards or leftwards in certain situations.

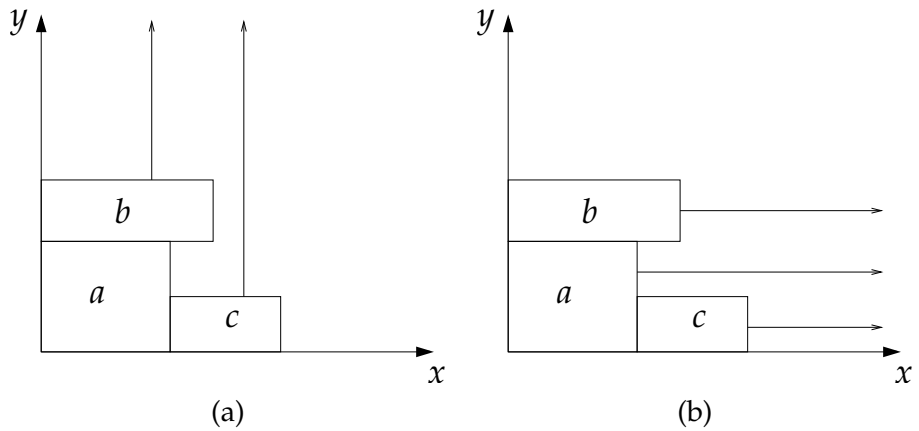


Figure 4.21: *Visibility. (a) A rectangle is said to be visible from above if there exists an infinite vertical line from the upper edge of the rectangle towards infinite y which does not intersect any rectangle. In this case rectangles b and c are visible from above and a is not. (b) Visibility from right is defined similarly. A rectangle is visible from right if there exists an infinite horizontal line from the right edge of the rectangle towards infinite x. All three rectangles are visible from the right.*

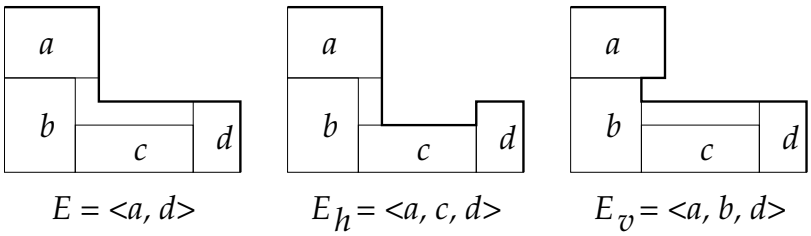


Figure 4.22: *An intermediate placement with envelopes defined by E, E_h and E_v. The envelopes are illustrated with thick lines.*

Since visible rectangles are extreme rectangles we now have three sets of extreme rectangles; E, E_h, E_v. Note that since any rectangle in E must be visible from both above and right $E = E_h \cap E_v$. Envelopes defined by the three sets are illustrated on figure 4.22.

Placement of a rectangle The placement of a rectangle h is similar to that of [72]. The algorithm is described in algorithm 4.5.

Based on figure 4.23 the algorithm proceeds as follows. First the predecessor a and successor g of h in E is determined as in algorithm 4.4. This defines the intermediate position of h between a and g.

At this point we decide whether we wish to attempt to "slide" h down or left. Assume that we wish to slide h down. h's x-coordinate is now locked.

Since we wish to move h downwards we update E_h first. We have determined h's

x-coordinate based on a's right side. We now determine a' successor in E_h, c, and traverse E_h right until we reach a rectangle, f, that remains visible with h's current x-coordinate. Every rectangle which becomes invisible from above by the addition of h in E_h is concurrently deleted from E_h. While traversing E_h rightwards we also determine the top most feasible placement of h by looking at the rectangles of E_h. This defines the y-coordinate of h. Determining a's successor in E_h can be done in constant time, by including a reference from every rectangle in E to its position in E_h and E_v.

Algorithm 4.5: Extended envelope sequence-pair placement

Input(*Sequence-pair:* $A =< a_1, \ldots, a_n >, B =< b_1, \ldots, b_n >$);
$E =< s, t >$;
$E_h = t$;
$E_v = s$;
for $k = 1$ ***to*** n **do**
  $h = b_k$;
  Determine p and q as in algorithm 4.4 ;
  Decide horizontal or vertical movement based on horizontal and vertical distance between p and q;
  if *Vertical Movement* **then**
    Find p in E_h ;
    $x(h) = x(p)+p_w$;
    Traverse E_h from p's position until a rectangle m not shaded by h is reached (i.e. $x(m) + m_w > x(h) + h_w$). Delete all rectangles shaded by h while traversing;
    Set $y(h) = y(s) + s_h$, where $y(s) + s(h)$ is the highest coordinate among the visited rectangles;
    $E_h = E_h \cup \{h\}$;
    Find q in E_v;
    Traverse E_v from q's position until a rectangle m not shaded by h is reached. Delete all rectangles shaded by h while traversing;
    if *h still visible from right* **then**
      $E_v = E_v \cup \{h\}$;
  else
    Do as in the horizontal case, but with respectively x and y and E_h and E_v interchanged.
  Remove rectangles shaded by h from E;
  if *h inserted in both E_h and E_v* **then**
    $E = E \cup \{h\}$;

Having determined the y-position of h we proceed to update the vertical envelope E_v and finally the regular envelope E. Both are updated according to the final location of h. However if h is not visible from the right at its final location it not inserted into

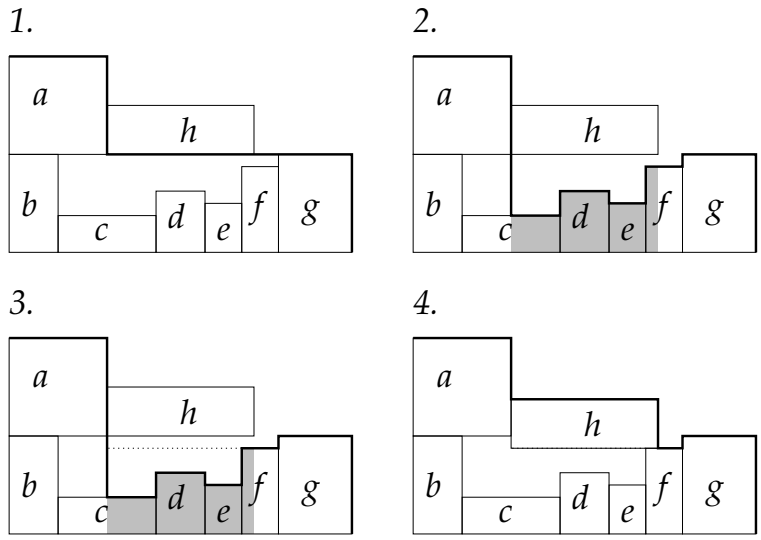


Figure 4.23: *Updating the horizontal envelope $E_h =< a, c, d, e, f, g >$. 1. The regular envelope E is used to determine the predecessor a of h. This gives the x-position of h and an intermediate y-position. 2. a is used to determine the reference into E_h. a's successor in E_h is the first rectangle horizontally shaded by h. E_h is traversed from c rightwards until a rectangle, f, not shaded by h is reached. 3. The y-coordinate of the top most edge of the visited rectangles is used to determine the final y-coordinate of h. 4. While traversing E_h rectangles shaded by h are removed from E_h and h inserted between a and f giving the new horizontal envelope $< a, h, f, g >$.*

E_v and E. Updating E_v at this point is similar to E_h as described above except the rectangle placements of E_v are not used to determine the x-coordinate of h. Updating E is similar to [72].

Leftwards or downwards "slide" In the previous example E_h was used to determine the lowest feasible y-coordinate but alternatively E_v could be used to determine leftmost feasible x-coordinate.

It is however not clear how to determine which direction results in the better placement. As a prime objective we consider most compact placements. We have decided to let the rectangles of E determine the direction. We consider the horizontal and vertical gap between the two extreme rectangles of E which determine the intermediate position of the rectangle to placed. If the horizontal gap divided with the width of the rectangle is larger than the vertical gap divided by the height of the rectangle it is assumed that the placement will become more compact by a downwards movement. This is merely a heuristic solution towards compact placements and many similar approaches could be used.

For VLSI-placement one could choose the direction that would presumable lead to minimum netlength by counting the number of nets the module would be extreme of (similar to the prefer method mentioned in section 3.2.1). However tests with this

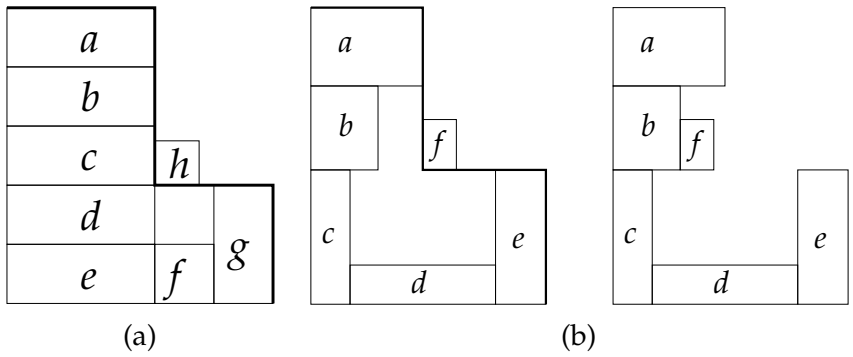


Figure 4.24: *(a) The heuristic determination of downwards or leftwards movement will attempt to move rectangle h leftwards instead of downwards since the vertical gap between a and g is larger than the horizontal one. Thus the best move is missed. (b) Only movement in one direction is possible. So in this case the rectangle f is still placed in a "floating" position.*

strategy showed worse results in terms of total netlength of the placement. The primary reason seemed to be that the placements were less compact, which increased overall netlength.

Asymptotic Running time The running time of the extended algorithm is $O(n \log \log n)$ where n is the number of rectangles. Determining the predecessor of a rectangle in E and updating E is as in [72]. Updating the two other envelopes E_h and E_v is done in amortized constant time for each rectangle. This follows from the fact that updating E_h and E_v deletes any rectangle visited except one (the last). Thus updating the two additional lists takes a total of $O(n)$ time. The running time is dominated by searching and updating the van Emde Boas Trees and is thus $O(n \log \log n)$.

Short comings The extended envelope algorithm as described above has two immediate short comings.

- Determining leftwards or downwards "slide" heuristically as described above does not always lead to a *better* solution.
- If both a leftwards and a downwards "slide" is possible only one is done.

The two short comings are illustrated on figure 4.24.

Fixed rectangles VLSI-placements in general may contain fixed modules. When placing a rectangle r_0 at a corner c using the algorithms we may place it on top of a fixed rectangle. Assume for now that we have a way to determine if this happens. At this point we insert the fixed rectangle r in the main envelope. We search for the

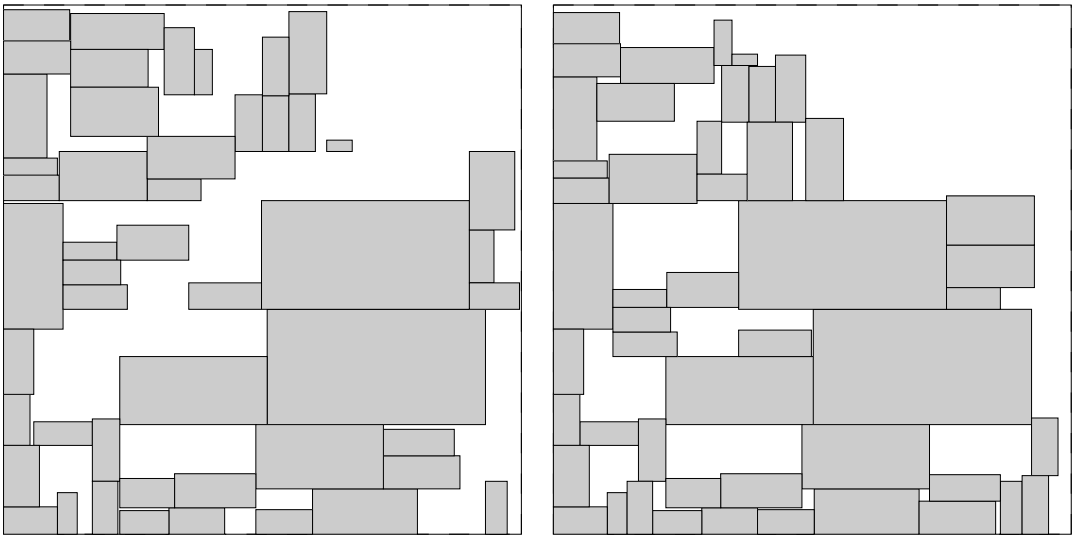


Figure 4.25: *Example of the extended envelope method. (a) Normal envelope sequence-pair placement. (b) Extended envelope placement.*

top-most corner below r and insert r at this corner. All corners between the corner of r_0, c, and this corner will be shaded since otherwise r would have overlapped with a rectangle in a previous step (see figure 4.26).

Inserting the fixed rectangle r with lower-left coordinates $(x(r), y(r))^t$, width r_w and height r_h at $(x', y')^t$ can be done by inserting a rectangle of size $(x(r)-x'+r_w, y(r)-y'+r_h)^t$ at $(x', y')^t$ in the main envelope E (see figure 4.27(a)). When r has been inserted we restart the insertion step of r_0.

It is slightly more complicated for the two auxiliary envelopes E_h and E_v. Here we need to insert dummy rectangles of zero height and width respectively so that the

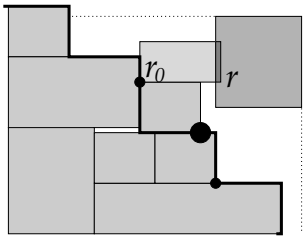


Figure 4.26: *Determination of corner for fixed rectangle. Assume a rectangle r_0 is to be inserted. When we insert r_0 we discover that it collides with a fixed rectangle r. Now we wish to insert r. We search in the envelope for the top-most corner below r (large circle). Any corner between the original corner and this top-most corner must be shaded by r since r_0 overlaps with r otherwise another previous rectangle would have collided with r.*

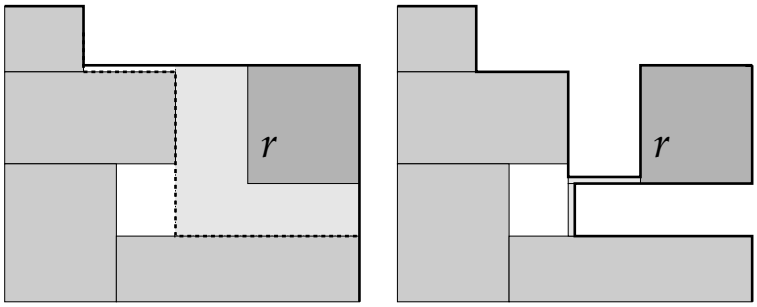


Figure 4.27: *Handling fixed rectangles in the envelopes. (a) If a fixed rectangle r needs to be inserted in the main envelope E a dummy rectangle of slightly larger size is inserted. (b) To handle the two other envelopes two dummy rectangles are inserted so that E_h and E_v do not allow for rectangles to slide too far down behind r and create inconsistency.*

envelopes do not become inconsistent with placements (see figure 4.27(b)).

Determining whether a collision happens during the placement of r may easily be done by a two-dimensional search trees with search time $O(\log n)$. However this increases the running time of the placement algorithms to $O(n \log n)$ which is still acceptable.

Limited placement area The VLSI-placement benchmarks have limited size. In this case we should avoid placing rectangles outside the placement area. This complicates the placement algorithm. We can solve the problem with two search trees. One is indexed according to the y-coordinate of the top of the rectangles in E. The other according to the x-coordinate of the right side of the rectangles E. Whenever rectangle h is to be placed outside the placement area we use one of the search trees to determine the position of h. So if e.g. h would be above the placement area we determine the first legal coordinate of h by using the rectangle-top search tree (see figure 4.28(b)).

A problem with this method is that it messes with the ordering of the $\mathbf{A}$ sequence so simply using the rectangle of $e_i \in E$ with highest index in $\mathbf{A}$ in algorithm 4.4 may return a poor location (see figure 4.28(c)) To solve this problem we mark rectangles which have been moved into the placement area. If e_i is marked, we place h as high or right as we can without violating the placement area using the search trees (see figure 4.28(d)) instead of using e_i to determine the location for h.

The argument for placing h as high or right as we can is that if e_i fell outside the envelope there is a good chance that h would also, and while the relation between h and its predecessor as specified by the sequence-pair is not maintained, relations between h and other rectangles may be. As an example of this, reconsider figure 4.28. Here d is supposed to be above a and b. Unfortunately c is moved into the placement

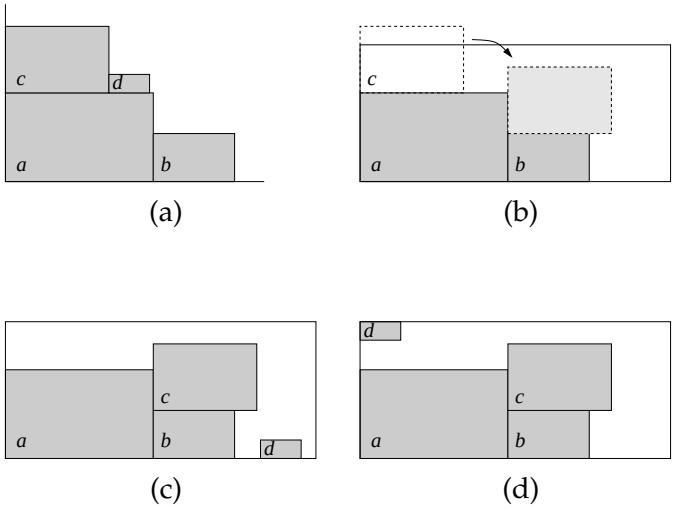


Figure 4.28: *Four rectangles with the sequence-pair* $\langle < c, d, a, b >, < a, b, c, d > \rangle$ *are to placed inside a limited placement area. (a) The placement without limitations. (b)* c *is moved down to the first legal position. (c) Moving* c *and using the algorithm directly would position* d *here. (d) Instead we observe that* c *has been moved down and position* d *as far up as we can.*

area so placing d with c as predecessor would maintain d's relation to c but not with a and b. On the other hand if d is placed as far above as possible it will still be above a and b.

Unfortunately the search trees require insertion and searching times of $O(\log n)$ increasing the running time of the placement algorithm to $O(n \log n)$ total.

Row-packing and cell-spacing VLSI-instances may contain large blockages mixed with standard-cells. To To ensure that modules are packed in rows we re-scale the large blockages so their bottom and top are aligned with a row. Pins are also realigned to fit the new size. If all modules have height equal a multiple of a row-height then the final placement will also be row-based.

To deal with spacing we simply expand modules during the sequence-pair placement.

P-admissible Without the limited placement area the algorithm guarantees P-admissibility since all codes are feasible placements even with fixed rectangles.

Comparison of the Standard and Extended Envelope Methods It seem natural to compare performance of the extended envelope method with the standard envelope method for area minimization. In section B.1 we describe how we have done this in

a simulated annealing framework. The conclusion is that the difference in quality is only noticeable during the first steps of the combinatorial search. At later stages the placement is presumably compact enough that the extended envelope makes little difference. In general the two methods perform almost equally. It is likely that the extra run time spend searching for better compactions of the standard envelope method is equal to the extra run time induced by the higher constants of the more complex extended envelope method. It should also be noted that our results of both methods are among the best ever reported, which demonstrates the strength of envelope-based placement.

4.5 The Legalization Algorithm

We are now ready to combine the pieces of the previous sections to form the legalization algorithm. The outline of the complete algorithm looks as follows:

1. Remove overlap by cutting with algorithm 4.3.
2. Convert non-overlapping placement to sequence-pair $\langle \mathbf{A}, \mathbf{B} \rangle$ with algorithm 4.2
3. Convert sequence-pair to a semi-normalized placement using algorithm 4.5 with extensions.

Figures 4.29 and 4.30 illustrate the algorithm on the circuit ami49. The result of the cutting step is shown on figure 4.31.

4.5.1 Centered Legalization

The algorithm can easily legalize a VLSI placement but this comes at a cost; the rectangles will be semi-normalized towards the lower-left corner of the placement area. In some cases this is not optimal since connections between modules and IO-pads on the upper and right boundaries will be very long. Secondly the more rectangles the harder it is for the legalization algorithm to legalize without large movement. Therefore we propose a legalization in the center of the circuit.

This is achieved by dividing the rectangles into four sets, R_{ll}, R_{ul}, R_{lr}, R_{ur} and placing the four sets individually. The sets contain the rectangles of the upper-right, lower-right, lower-left and upper-left part of the placement area respectively. Now the modules of the four sets are placed in four equally sized quadrants of the placement area. All four placement have the center of the circuit as origin but such that the upper-right set is placed towards upper-right corner of the placement area, the lower-right rectangles are placed towards lower-right corner of the placement area etc. The placement method is illustrated on figure 4.32.

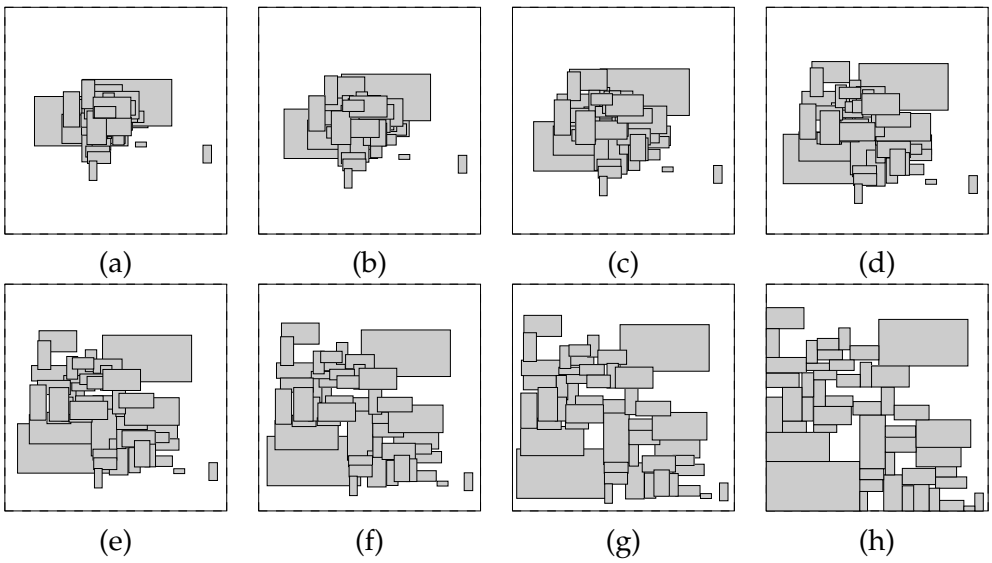


Figure 4.29: *Animation of the legalization algorithm on ami49. (a) The original placement based on unconstrained quadratic placement with massive overlap. (h) The legalized placement. (b)-(g) movement of modules. These images are interpolated placements between overlapping and legalized placements and are not in any way related to the algorithm. Their sole purpose is to illustrate how modules are moved during legalization.*

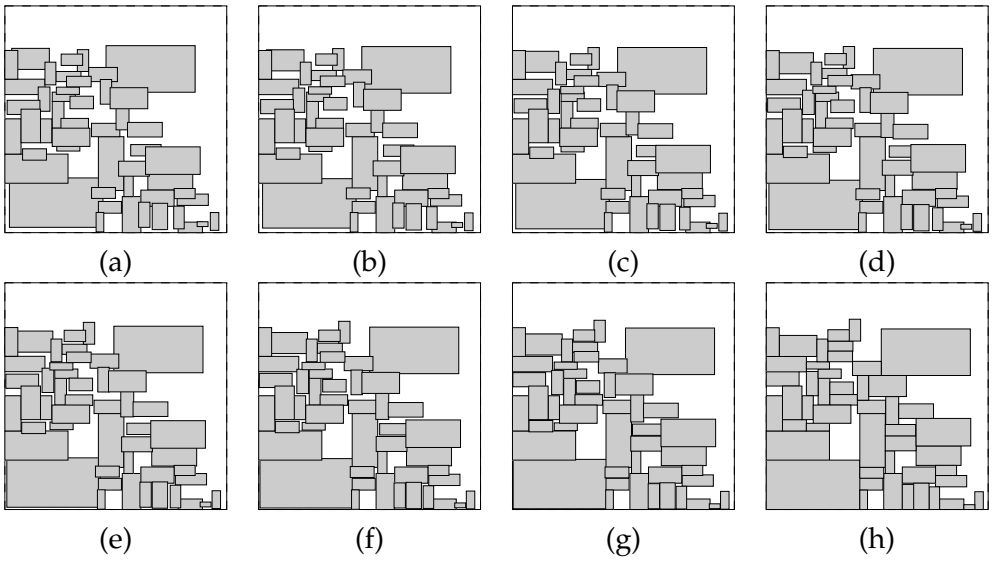
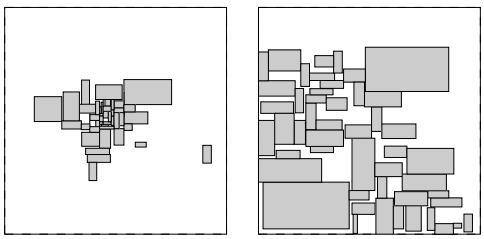


Figure 4.30: *Animation of the legalization algorithm on ami49. (a) The original placement with some overlap. (h) The legalized placement. (b)-(g) movement of modules. These images are interpolated placements between overlapping and legalized placements and are not in any way related to the algorithm. Their sole purpose is to illustrate how modules are moved during legalization.*



(a) (b)

Figure 4.31: *Results of the overlap-removal by cutting algorithm. (a) Cutting result of legalization from figure 4.29. (b) Cutting result of legalization from figure 4.30.*

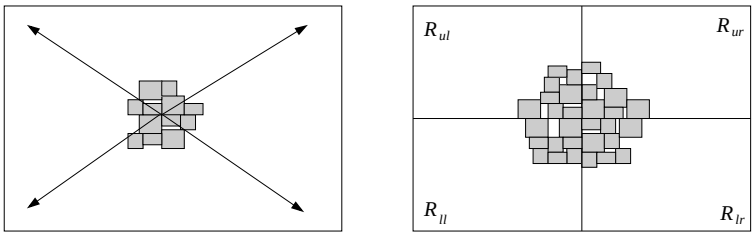


Figure 4.32: *Center placement. Rather that placing the rectangles with the lower left corner of the placement area as origin the rectangles are divided into four groups which are each placed with the center of the circuit as origin but with their placements builded towards lower-left, lower-right, upper-left and upper-right corners of the placement area.*

We use two different strategies to divide the modules into the four sets.

The first strategy we refer to as *simple* centered legalization. Here the placement are is divided into four equal-sized rectangular regions corresponding to the four mentioned set. A rectangle is placed in a set if its center belongs to the region of the set.

The second strategy we refer to as *complex* centered legalization. Let R be the set of rectangles and $P = (x, y)$ be the current placement. To determine the four subsets of R we partition R in the following manner:

1. Sort the rectangles of R by x-coordinate.
2. Let r_a equal the area of a rectangle from R. Then create subset $R_l \subseteq R$, by adding modules from the sorted set R in left to right order until $\sum_{r \in R_l} r_a \geq \frac{1}{2} \sum_{r \in R} r_a$. Let $R_r = R \setminus R_l$.
3. Sort the set R_l by y-coordinate. Create a subset $R_{ll} \subseteq R_l$ by adding modules from the sorted set R_l in bottom to top order until $\sum_{r \in R_{ll}} r_a \geq \frac{1}{2} \sum_{r \in R_l} r_a$. Let $R_{ul} = R_l \setminus R_{ll}$.
4. Create similar sets R_{lr} and R_{ur}.

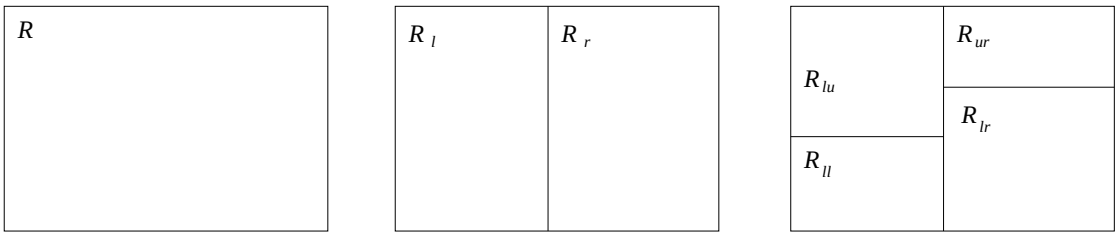


Figure 4.33: *The sets of the four-partition step.*

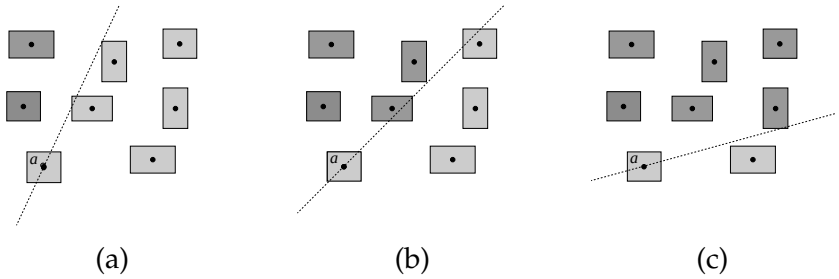


Figure 4.34: *Effect of α. (a) Slope of the diagonals is high. Few rectangles are considered above a in the placement-to-sequence-pair algorithms. (b) and (c) Slope is lower more rectangles will be considered above a.*

The sets R_{ll}, R_{ul}, R_{lr}, R_{ur} constitute a four-partition of R as shown on figure 4.33.

We will use the two strategies differently. For placements with massive overlap it can be hard to place the modules within the placement area if e.g. one of the four groups of the simple method contain the majority of the modules. Therefore we divide modules almost evenly among the groups.

For placements with little overlap the simple method work best since a module will lie in the same region on the placement area before and after legalization.

4.5.2 The α-Parameter

The α-parameter of algorithms 4.1 and 4.2 can affect the legalized placement. The α-parameter describes the slope of the diagonals in the placement to sequence-pair algorithms. Consider the placement of figure 4.34. The lower the slope is the fewer rectangles will be considered above the rectangle a in the placement to sequence-pair algorithms. The higher the slope the more rectangles will be considered right of a.

This allows us to "rotate" the legalized placement to distribute the modules more evenly on the placement area after legalization. Figure 4.35 shows how different α-values affect the legalized placement of a real-life circuit. Needless to say the right choice of α may effect the quality of legalized placement.

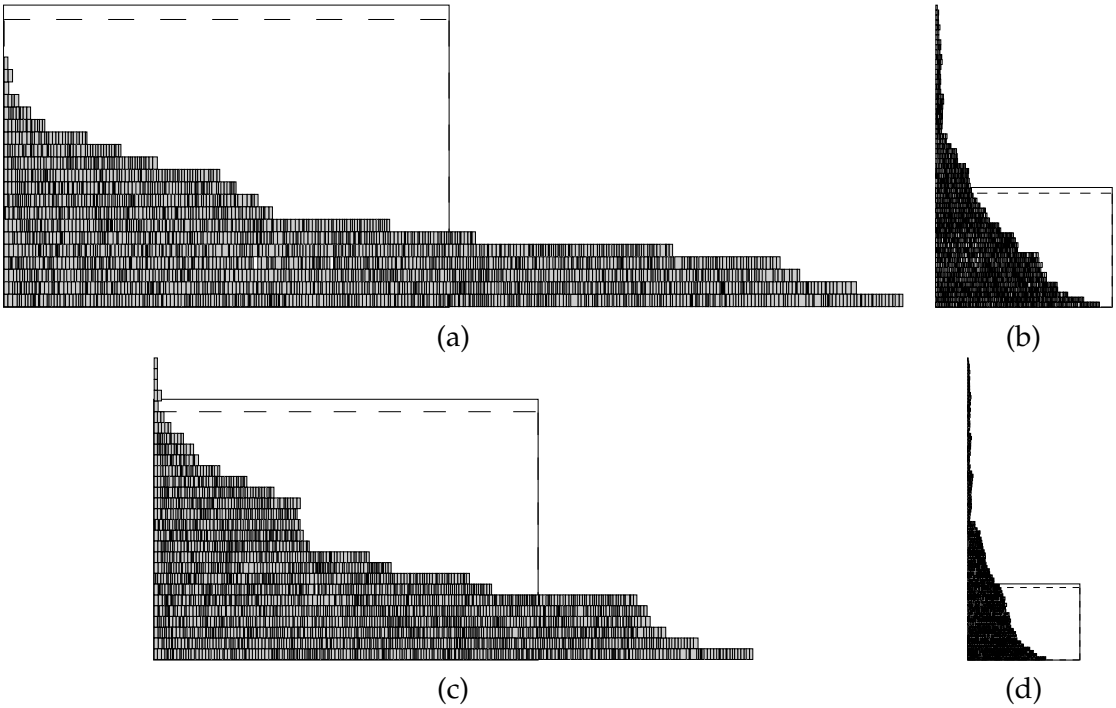


Figure 4.35: *Different legalized placements of the industry1 standard cell circuit. The placement is not limited to any specific placement area, however the placement area is shown. Notice that higher values of α makes the placement "taller". (a) $\alpha = 0.025$. (c) $\alpha = 0.05$. (b) $\alpha = 0.2$. (d) $\alpha = 0.4$.*

4.5.3 Remarks on Netlength Considerations During Placement

Although it would seem natural to consider the netlength during placement all our attempts have proven otherwise.

We have tried the following strategies during the placement algorithm:

- **Smallest netlength contribution in envelope** Instead of choosing the position for a module based on its index in the **A**-sequence we tried choosing the position which minimized netlength. The netlength was measured such that for modules not placed yet their illegal position was used. In general this worked worse than straight-forward approach. This could be explained by the fact that the relative module position is not maintained in the placement.
- **Smallest move** We also tried to place modules at the position closest to their illegal coordinates. This approach failed completely with placements with great amounts of overlap and was in general not better than the strategy based on the **A**-sequence. This strategy probably has the same flaw as the previously mentioned one.
- **Slide in best direction** Finally we tried to slide modules in the direction which was likely to minimize netlength (based on "prefer"-values) instead of the direction with largest hole. The problem with this strategy was that the placement would be less compact and therefore have longer netlength.

Based on preliminary results all three strategies were abandoned.

4.5.4 Poorly Legalized Placements

Although the legalization can perform well in some cases it will perform poorly in other cases where modules relative order is altered. This can happen for a number of different reasons:

- The legalizer does not consider module distribution on the placement area. Also if modules overlap severely the cut algorithm will distort sizes. Therefore the placement algorithm will build placements unevenly which will distort the internal module relations, since modules are also pushed as far left and down as possible.
- The legalizer assumes that modules can fit within the placement area. This is also not the case. Often the legalizer has to fix placements that violate the placement area. This affects the internal module relations a great deal.

- Overlap with fixed modules is not allowed. Therefore the placement algorithm moves modules beyond the fixed modules, but this also alter the internal module relations.

Many preliminary attempts at dealing with these problems failed during this work. We experimented with placement-to-sequence-pair algorithms that considered module sizes and we tried to redistribute modules evenly on the placement area before converting to sequence-pair. None of these methods improved placements noticeably and in most cases they functioned more poorly than the method we have described in the previous sections. The unsuccessful approaches towards dealing with these problems are listed in more detail in appendix A

5 Benchmark Circuits

Before discussing the layout problem further we will briefly consider the standard layout circuit benchmarks. The purpose of introducing the benchmarks at this point is to give the reader a better understanding of the complexity and structure of the VLSI layout instances. In the first half of this section we will discuss the benchmark characteristics and in the final half we will propose three new general-cell benchmarks based on common benchmarks.

5.1 About the Circuits

We will consider three kinds of circuit instances. The MCNC macro-block instances, the MCNC standard-cell instances and three IBM real-life circuits.

5.1.1 MCNC Macro-Blocks

The MCNC macro-block instances are small compared to the standard-cell circuits. The largest of the five instances contain only 49 modules and about 400 nets. The instances comes from the 1991 Physical Design Workshop [50]. These benchmarks have been used to demonstrate area minimization. The modules represent large macro-blocks. The macro-block instances are called: apte, xerox, hp, ami33 and ami49.

5.1.2 MCNC Standard-Cells

The MCNC standard-cells were also released at the Physical Design Workshop 1991 [50]. All cells have equal height. Writers have not agreed on spacing between rows. Most writers use row-spacing and horizontal spacing between cells although recent papers have abandoned the spacing due to advances in technology[9]. The MCNC standard-cells circuits vary from a few hundred modules and nets to almost 100.000 modules and 150.000 nets. The standard-cell instances are called: industry1, industry2, industry3, primary1, primary2, avqlarge, avqsmall struct and fract. The fract circuit contains only 125 modules and is rarely used in articles because of its small size.

Although not a MCNC-benchmark circuit, the golem3 circuit is commonly used by writers in conjunction with the other benchmarks.

[9] As a side note: The lack of row spacing can improve netlength substantially.

5.1.3 IBM Real-Life Circuits

The final type of circuits we will describe is the IBM circuits. These contain cells of various width and *height* however the majority of the cells are of equal height. There are only few macro-blocks. Further the design rules restrict placement of modules to rows of specified height. The IBM instances contain as many as almost 170.000 modules and 190.000 nets. To further complicate matters these instances also contain a number of fixed modules. The number of fixed modules is listed in table 5.2. The IBM instances are called clk, decoder and pu. At this point we would like thank IBM and Associate Professor Martin Zachariasen for supplying us with the real-life circuits.

5.1.4 Data Format and Origin

The MCNC standard-cell benchmark instances we use are not the original MCNC-benchmarks, but are modified instances originating from P. Madden comparison of standard-cell placers (see section 3.3 and [57]). The instances were modified in [24] and later converted to an XML-based file-format in [25], which is the format we use. The modifications of [24] were mainly with respect to pin-positions and size of placement areas. The MCNC-specifications does not specify size of placement area so the size used by [24] is based on test-results conducted by P. Madden [57] on the TimberWolf-placer. However Færø increased the width of the placement area by 20% in [24] so effectively there is about 20% surplus space on the placement area. See [24] for the exact modifications of the circuits.

We have decided to use the benchmarks from [24] instead of the original MCNC-benchmarks because it allow us to do an honest comparison of our results with those of [24] since circuit-size, pin-locations and row-height are identical.

5.2 About the Circuit Data

Various characteristics of the circuits are listed in table 5.1.

- **Number of modules and nets are roughly equal** It seems that the number of modules and the number of nets are of same order disregarding the macro-block instances. I.e. 100.000 modules implies in the order of 100.000 nets.
- **Average number of pins is low**. The average number of pins is low. On average modules and nets contain as little as 3-4 pins disregarding the macro-block instances. For the macro-block instances nets still contain very few pins.
- **Modules are of almost equal size** Not surprisingly the standard-cell instances contain only few different sized modules. What is more surprising however is that the real-life circuits from IBM also contain few modules of different size.

Instance name	Modules	Nets	Module sizes	Area	Module Pins	Net Pins
apte	9	97	3	-	23.78	2.21
xerox	10	203	10	-	69.60	3.43
hp	11	83	6	-	24.00	3.18
ami33	33	123	31	-	14.55	3.90
ami49	49	408	47	-	19.00	2.28
industry1	2271	2478	25	-	3.53	3.24
industry2	12142	13419	17	-	3.95	3.57
industry3	15059	21938	13	-	4.52	3.10
avqlarge	25114	25384	5	-	3.29	3.26
avqsmall	21854	22124	7	-	3.49	3.44
biomed	6417	5742	7	-	3.26	3.65
fract	125	147	7	-	3.5	3.14
struct	1888	1920	4	-	2.90	2.85
golem3	99932	144949	20	-	3.36	2.32
primary1	752	901	8	-	3.73	3.12
primary2	2907	3028	8	-	3.80	3.65
decoder	54930	59256	38	58.2 %	3.37	3.12
pu	164510	184231	90	65.6 %	3.75	3.35
clk	29410	30293	36	19.1 %	3.79	3.68

Table 5.1: *Data for the benchmarking circuits. The "module pins" and "net pins" numbers are respectively the average number of pins on each modules and in each net. The area value is the placement area divided with area of the modules. I.e. placed legally the modules will cover this percentage of area not including required spacing. No placement area is specified for the MCNC-benchmarks.*

Instance	Fixed modules
decoder	19
clk	354
pu	550

Table 5.2: *Number of fixed modules of the IBM real-life instances. Even for these very large instances the number of fixed modules is low.*

- **Few fixed modules** The IBM real-life circuits contain few fixed modules. Even the largest of the instances contain only 550.

The observations are important because a heuristic may exploit some of these observations. E.g. the fact that modules are of equal size or that the number of nets connecting each module is low.

5.2.1 Pin Distribution

If we investigate the pin distribution further we get an interesting result. The pins per module distribution is plotted on figure 5.1. This shows than 80 % of the modules of the large instances contains ≤ 5 pins and less than 5 % contain more than 10 pins. Looking at figure 5.2 we see the same pattern for nets. 80 % of the nets contain less than 5 pins and less than 5 % contain more than 50 pins. This is interesting because it means that the underlying hypergraph is sparse.

5.2.2 Size Distribution

We also consider the distribution of the module sizes. Figure 5.3 shows this distribution on each of the three types of circuits.

It is interesting that the standard-cell circuits contain modules of sizes between 1 and 10 times the smallest module. For the real-life circuits 80 % of the modules are smaller than 10 times the smallest module and more than 95 % of the modules are smaller than 25 times the smallest module. This demonstrates that the real-life circuits consists mainly of standard-cells. It should also be noted that the real-life circuits contain modules of sizes up to 75.000 times the smallest module. However some of the largest modules are fixed modules.

The size distribution tells us that a heuristic should probably be geared towards modules of close to equal size.

5.3 New Benchmarks

In this section we propose new benchmarks. We do this because our placement heuristic is capable of placing general-cells and the benchmarks of the previous sections are all standard-cell benchmarks – although the IBM circuits do contain macros.

Since we wish to compare results of our heuristic with others we will base the new circuits on existing ones. We will use two approaches:

- **Tiling of macro-block circuits.** A macro-block instance is tiled creating $n \times m$ copies of the original circuit. Modules connected to IO-pads of the original

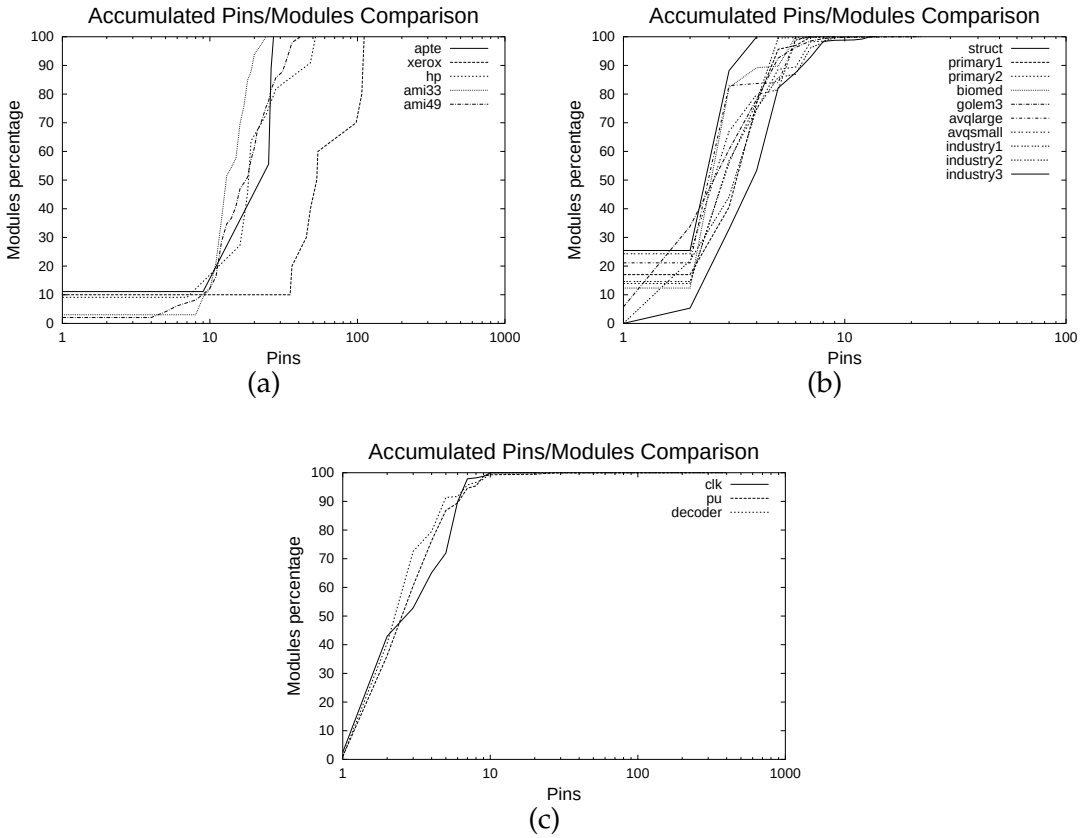


Figure 5.1: *Pins on modules. Percentage of modules (y-axis) containing* less *than or equal ($\leq$) the given number of pins (x-axis). (a) macro-block instances. (b) standard-cell instances. (c) Real-life circuits.*

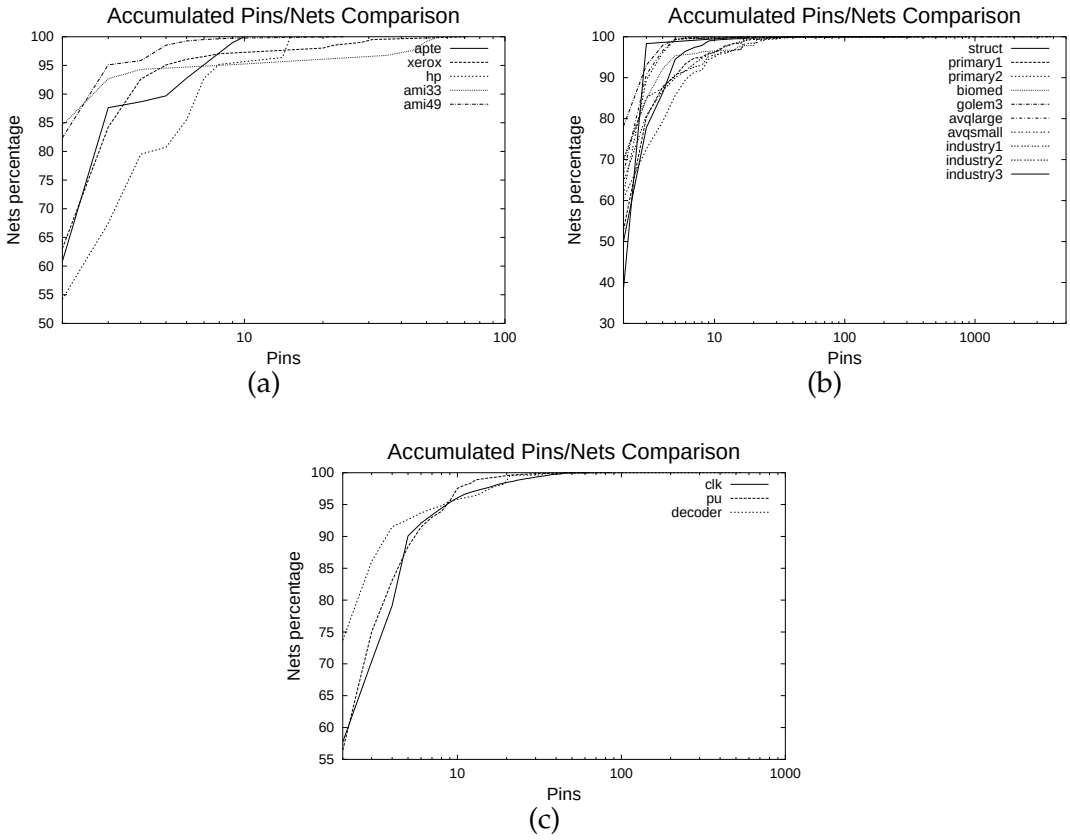


Figure 5.2: *Pins on nets. Percentage of nets(y-axis) containing* less *than or equal ($\leq$) the given number of pins (x-axis). (a) macro-block instances. (b) standard-cell instances. (c) real-life circuits. (x-axis starts at 2)*

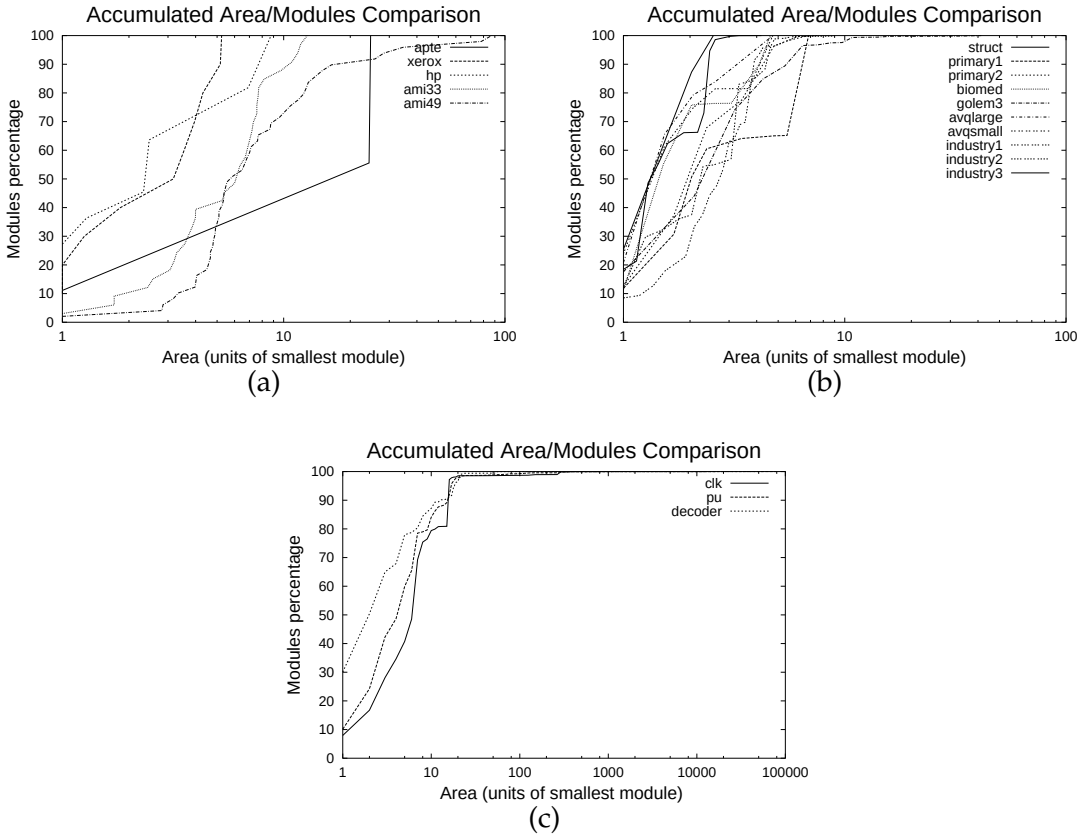


Figure 5.3: *Size distribution of cells. Percentage of modules (y-axis) smaller than or equal (leq) the area (x-axis). The area is in units of the smallest module. (a) macro-block instances. (b) standard-cell instances. (c) real-life circuits.*

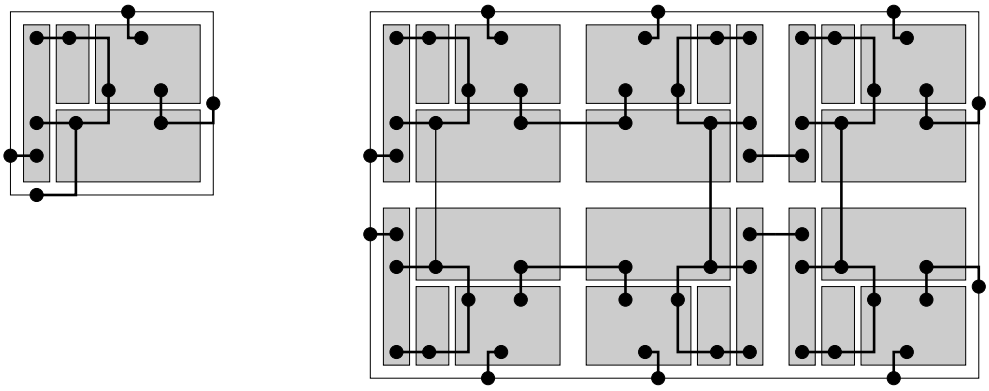


Figure 5.4: *Tiling of a circuit to create a new larger instance. Left is the source circuit. Right is a* 2×3 *tiling. Only IO-pads on the boundary of the destination circuit are maintained. Shared IO-pads inside the boundary are deleted and nets connecting these are merged.*

circuit are connected to modules of any neighboring copy instead (see figure 5.4).

- **Modifying standard-cell circuits.** The standard-cell instances are modified. A new instance with equal total module area is created, but the modules are scaled. For each module ratio r is randomly picked uniformly between 1 and 3, meaning that the new width w will be $w^2 = \frac{a}{r}$, where a is the area of the original module. Based on the new width the new height can be calculate ($h = \frac{\sqrt{a}}{\sqrt{r}}$). Pin offsets are scaled according to the new dimensions of the module.

In the first case we hope that the resulting optimal netlength may be close to $n \times m$ times the optimal netlength of the original macro block circuit. Of course it is not unlikely that the netlength may be reduced in the large instance since the increased space allows for new packings of the modules.

In the second case we hope that by not changing the area of the modules the optimal netlength will be more or less the same. Of course new packing configurations are allowed resulting in possible lower netlength. On the other hand general-cells makes it harder to compact the modules for low netlength without creating unused areas.

We have created three circuits based on these two methods: *ami33K* which is a tiling of $32 \times 32 = 1024$ instances of ami33 and *primary2g*, *industry2g* and *industry3g* which are general-cells versions of the popular industry2 and -3 benchmark circuits. The ami33K circuit contains the ami33 circuit 1024 times, which means that it contains 33792 modules and 115392 nets, making it a very tough instance. Ami33 was chosen because it contains many differently sized modules (31) and only four times as many nets as modules.

In order to determine the placement area for ami33K we used the packing results of appendix B of ami33. We scaled the horizontal and vertical dimensions of the minimal

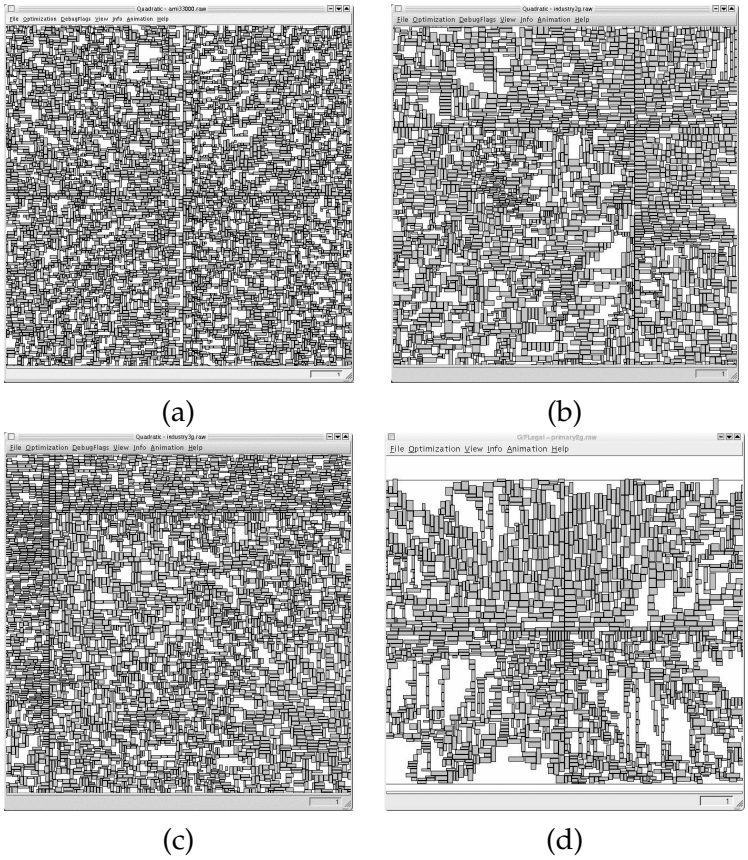


Figure 5.5: *Extracts of the three new instances. (a) ami33K. (b) Industry2g. (c) industry3g. (d) primary2g. The extracts are from legalizations of initial quadratic placements and are screen-grabs of our placer (see section 9.1).*

packing by 32 in each direction corresponding to the 32×32 tiling. To compensate for the complications involved with packing the circuit we increased the width and height by 20%.

For industry2g and industry3g we expanded the height of the circuits by 20% to compensate for general-cell complications.

The three circuits are shown in figure 5.5.

A problem with ami33K which we have not dealt with is the connectivity of the tiled sub-circuits. The tiled sub-circuits are connected to each other but only weakly since there are relatively few nets connected to the IO-pads of ami33. The resulting relaxed placements therefore becomes a tiling itself (see figure 5.6). This simplifies global placement a great deal since the relaxed placement positions modules inside their respective tiles which is where one would expect them to be positioned in an optimal placement.

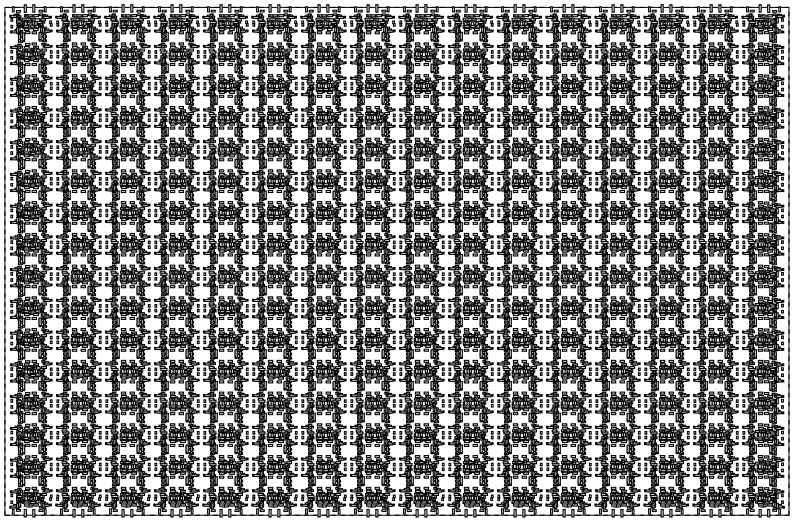


Figure 5.6: *Relaxed placement of ami33K (star netlength). Because of weak connectivity of the modules of ami33 and its IO-pads the relaxed placement becomes a tiling, which simplifies global placement.*

5.4 Rotations and Mirroring

According to [24] orientation and mirroring of the MCNC standard-cell and IBM real-life circuits are not allowed and therefore we will not permit this for these circuits in the placement heuristics of the following sections.

However for the new general-cell circuits we will permit rotation and mirroring.

6 Local Search for The Placement Problem

A central part of both our new global placement and final placement methods is a local search method which we will present in this section. The most common local search methods for final placement are:

- **One-move** Move a module to its optimal position with respect to length of incident nets.
- **Two-move (swap)** For a module determine the best exchange with another module with respect to the length of their incident nets.
- **Solve transportation problems** Several methods solve a form of transportation problem to determine where modules should move to.
- **Relaxed local search** Recent methods relax the no-overlap constraints of a subcircuit and solve the bounding-box formulation with maximum-flow algorithms.

See section 3.2 for more details of these methods.

We have decided to use the swap-based method for a number of reasons. Firstly it will not place modules directly on top of each other and is therefore less likely to produce highly overlapping placements. This is important since our legalizer can handle some overlap but has difficulties with severe overlap. Secondly it can be implemented relatively efficiently. Thirdly it is a much simpler method than the transportation- or relaxation-based methods.

Algorithm 6.1: Greedy swap-based local search for one module

Input(*A placement problem, and a module m_0 which should be optimized*) ;
$l_{\text{best}} = -\infty$;
$m_{\text{best}} = [\text{none}]$;
foreach $m \in \mathcal{M}$ **do**
    Swap positions of m and m_0 ;
    Let l be the netlength reduction after swap ;
    if $l > l_{\text{best}}$ **then**
        $l_{\text{best}} = l$;
        $m_{\text{best}} = m$;
    Swap m and m_0 back gain. ;
if $m_{\text{best}} \neq [\text{none}]$ **then**
    Swap positions of m and m_{best} ;
return *Improved solution*

In the following sections we will describe the local search method in detail. The outline of a simple swap-based local search for one module is shown in algorithm 6.1, however this straight-forward approach has two main flaws:

- Module sizes are not considered so swaps may result in placements with an uncontrolled amount of overlap, which could make it hard to legalize without moving modules too much.
- The potential positions are static; modules can only be placed at positions where there already is another module.
- Searching every other module of the circuit is computationally extremely expensive.

In the following sections we will address these flaws. In the next section we will consider how overlap can be accounted for during local search and also how new positions can be generated. In section 6.2 we show how the neighborhood can be reduced considerably and finally in section 6.3 we present our swap-based local search.

6.1 Overlap Handling

To ensure that the placement contains little overlap when modules are swapped we wish to consider the overlap resulting from a swap of two modules. Therefore we need to know the amount of overlap a swap results in.

6.1.1 Preliminary Considerations

Probably the most elegant method in conjunction with local search for the VLSI-placement problem for overlap-determination was developed by Færø et al. [24, 21]. Here the local search neighborhood consists of horizontal or vertical translations of one module m. The amount of overlap arising from a horizontal or vertical translation is determined by an efficient sweep-line algorithm which also determines bounding-box netlength. The sweep-line algorithm examines a subset of positions on the placement area.

Unfortunately swap-based local search presents us with a number of problems which renders this approach inefficient. Firstly, although preliminary investigation showed that it would be possible to implement an algorithm similar to the sweep-line algorithm efficiently there are two major complications.

- It must be determined which and how many modules to exchange position with. If more than one overlapping module is chosen then it must be determined how two or more modules should be placed at the original location of m. If only one is chosen then it must be determined which one.
- Also we deem that the one-dimensional search is insufficient for a swap-based neighborhood. If one would wish to do a two-dimensional translation it would

have to be broken into two one-dimensional steps which must both be accepted. The one-dimensional neighborhood made perfect sense in the guided local search framework, because only penalized overlap – i.e. overlap between certain pairs of modules – was considered which allows for more acceptances.

The answer to the second problem would be a two-dimensional translation algorithm. Our initial investigations show that it is possible to extend the one-dimensional sweep-line algorithm to two dimensions but it would become far more complex.

Because of these complications we have decided to calculate the overlap by an estimate instead. This makes sense since we are not interested in generating solutions with zero overlap. As long as the solutions have little overlap the legalization algorithm should be strong enough to remove it.

For each module we will let a rectangular region represent nearby area containing the module. This way we can determine if a module uses more than nearby space. Such a violation can be interpreted as overlap with neighboring modules. As the region is rectangular we should be able to determine this overlap estimate in constant time.

6.1.2 Pockets

To represent the surrounding area of a module we divide the placement area into a set of rectangular regions. Each such *region* we call a pocket. Pockets can contain modules but may be empty. Also pockets can be larger than the module they contain.

Traditionally division of the placement area occurs by bi- or quadrisection methods or by dividing it into a number of equally sized bins. In the spirit of this thesis we have decided to use the sequence-pair representation to divide the placement area by reversing the envelope-based placement algorithm. This is a novel technique for creating a floor-plan with cells containing each module and it is as simple to implement as the envelope-based placement algorithm.

First a sequence-pair for the current placement is determined by the placement-to-sequence-pair algorithm (algorithm 4.2 of section 4.3). Then the sequence-pair is placed in backwards order; the last module of the $\mathbf{B}$-sequence is placed first. This time we build a reverse envelope which contain the lower and left side of the modules. This is shown on figure 6.1.

Now as each module is placed we determine its index in the envelope using the $\mathbf{A}$-sequence just as in the placement algorithm but unlike the placement algorithm we do not use the envelope to determine the position of a module. Rather we use it to construct pockets. On figure 6.2 (a) we have shown how a module is inserted into the envelope. As the envelope is updated shaded modules are removed. While removing shaded modules from the envelope we can determine a pocket which contains the new module. This is shown on figure 6.2 (b). The remaining newly shaded area is

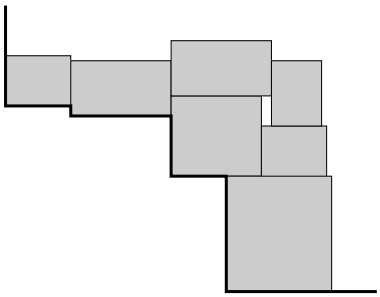


Figure 6.1: *Construction of pockets. An envelope is maintained containing the lower and left side of the modules.*

also divided into pockets as shown on 6.2(c). Finally the envelope is updated with the newly added module as shown on figure 6.2(d).

To determine the pocket for the module to be inserted we traverse the corners of the shaded envelope. This way we can determine the largest rectangle with respect to area that contains the module (see figure 6.3(a)). Any remaining area above or right of the created pocket is also divided into pockets. The area above we divide into rectangles between the top-side of the pocket and the envelope. The area to the right we divide into rectangles between the right-side of the pocket and the envelope (see figure6.3(b)).

An example of the result of the pocket algorithm is shown on figure 6.4.

Running Time The asymptotic running time of the pocket algorithm is equal to that of the envelope based sequence-pair-to-placement algorithm (algorithm 4.4) since all pockets are created by a constant number of extra amortized constant length envelope traversals; one to determine largest pocket, one to determine auxiliary pockets and one to remove shaded modules. All three traversals consider an equal number of modules from the envelope. Therefore the run time is $O(n \log \log n)$ as it was for algorithm 4.4.

6.1.3 Locations

Pockets allow us to determine a set of candidate locations for the lower-left corner of a module during moves in the swap-based local search.

A priori all pockets constitute a location. However even if a pocket contains a module m it may contain enough free space for it to be possible to position another module m' in it without causing too much overlap.

Based on these thoughts we use the following locations within a pocket p:

- A module can always be placed at the lower-left corner of a pocket.

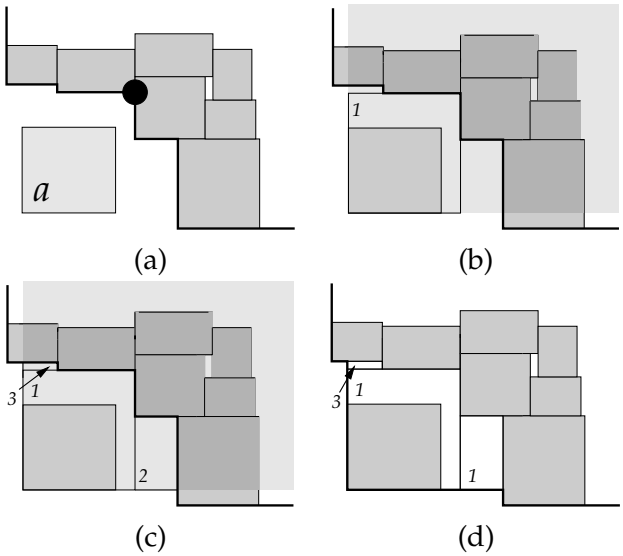


Figure 6.2: *Creation of pocket from placement. (a) Module a is to be inserted into the envelope. The black circle corresponds to its index in the envelope as determined from the* **A***-sequence. (b) The shaded envelope is traversed to determine a pocket for module a (1). (c) The remaining shaded area is also divided into pockets (2, 3). (d) The envelope is updated and the free-area surrounding a has been divided into three pockets.*

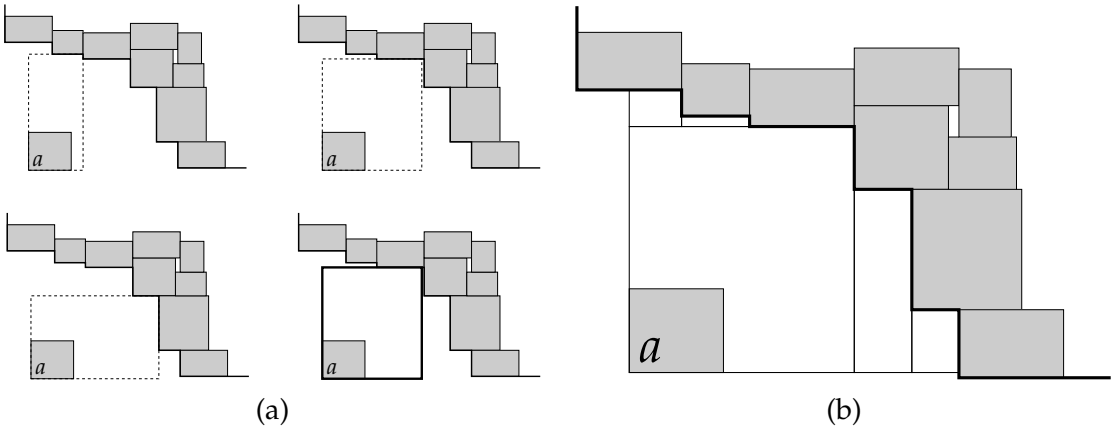
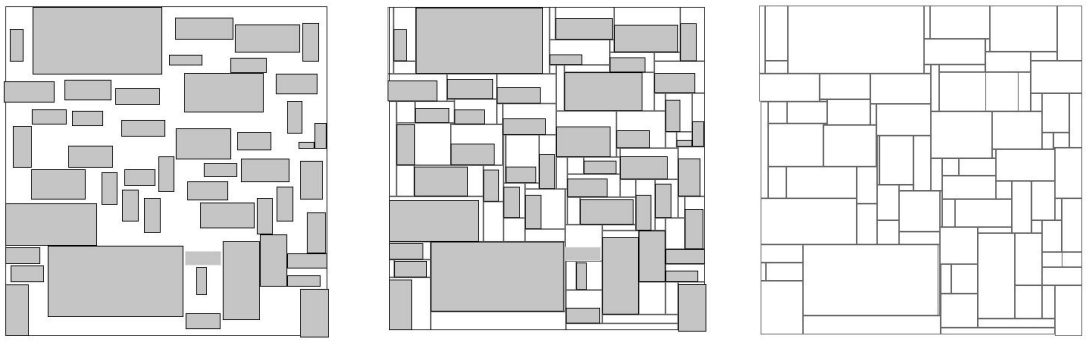


Figure 6.3: *Determination of pockets surrounding a module a. (a) The shaded envelope is traversed to determine the largest rectangle containing a. The three possibilities are tested. The lower-right figure shows the choice of pocket which is always the rectangle with largest area. (b) The remaining area is divided into pockets between the top of the pocket and the envelope and the right of the pocket and the envelope.*



(a) (b) (c)

Figure 6.4: *Result of the pocket algorithm. (a) Placement without pockets. (b) Modules and pockets arising from the placement. (c) Just pockets.*

- If a pocket contains a module m, then another module may be placed just right of m given that the pocket is strictly wider than m.
- If a pocket contains a module m, then another module may be placed just above m given that the pocket is strictly taller than m.

The three locations are depicted on figure 6.5. This way we can also position modules at new locations in the placement.

Pocket splitting and merging During local search we will not allow pockets to contain multiple locations, so if we use any of the two extra locations, we split the pocket in two pockets such that modules are contained in separate pockets. On the other hand a pocket may become empty in which case it may be possible to merge two pockets. This merging procedure may be simple but determining which other pocket to merge with seems difficult. Therefore we do no merging. This may lead to fragmentation but as will be explained in conjunction with our new global and final placement methods (see section 7.3 and 8.2.3) the placement area will be re-divided into pockets from scratch sufficiently often for fragmentation to be limited.

6.1.4 Augmented Objective Function

We introduced pockets to control the amount of overlap in the placement. It is very likely that there exists two modules of different size where a swap would improve the netlength. In this case the move may result in one of the modules being larger than its destination pocket. To handle this we penalize such moves. The penalty can be interpreted in two ways

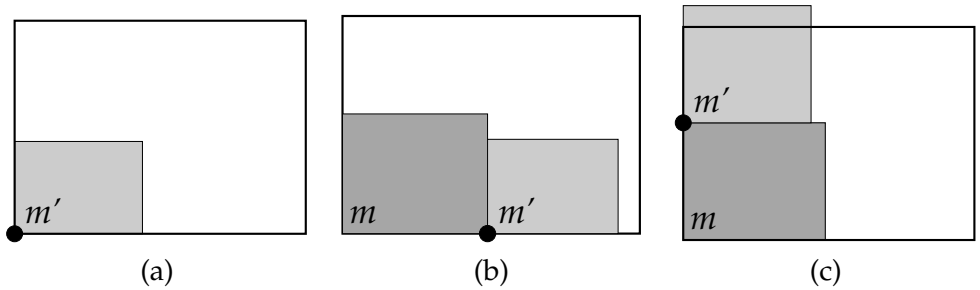


Figure 6.5: *Pockets may contain three locations for modules. (a) Any module m' can be positioned at the lower left corner of the pocket. (b) If the pocket contains a module m and is wider than m then another module m' may placed to the right of it. (c) If the pocket contains a module m and is taller than m then another module m' may be placed above it. Note that we may use a location even if a module is too large for it, since this is penalized in the objective function.*

- Overlap will move other modules during the legalization phase which is likely to increase netlength. Therefore the penalty should reflect the increased netlength due to legalization.
- Overlapping placements are a relaxation of the placement problem and should therefore be avoided in general. The penalty will make the local search avoid overlap if possible.

To penalize moves we introduce a penalty function for a given module at a given location. A location L has a free width L_w and free height L_h equal to the respectively the width of the free area right and the height of the free area above the location in the corresponding pocket (see figure 6.6). We now introduce a penalty function:

$$\text{sizepenalty}(L, m) = (\max(m_w - L_w, 0) + \max(m_h - L_h, 0)). \tag{6.1}$$

Note that the penalty should be interpreted as intersection-depth penalty (see definition 4.7) and not overlap-area penalty. The reason why we have chosen the intersection-depth strategy is that we expect it to better reflect netlength change due to legalization; modules are moved left and right a distance equal to intersection depths of overlapping modules.

During local search penalties are added to the objective function which will now have the form:

$$\min \sum_{n \in \mathcal{N}} w(n) \cdot BB(n) + \gamma \sum_{m \in \mathcal{M}} sizepenalty(L_m, m), \tag{6.2}$$

where $\gamma \geq 0$ and L_m is the location of module m. The constant γ allows us to control how relaxed the placement algorithm is. A small γ will allow moves with a lot of overlap, while a large γ will limit the amount of allowed overlap in the placement.

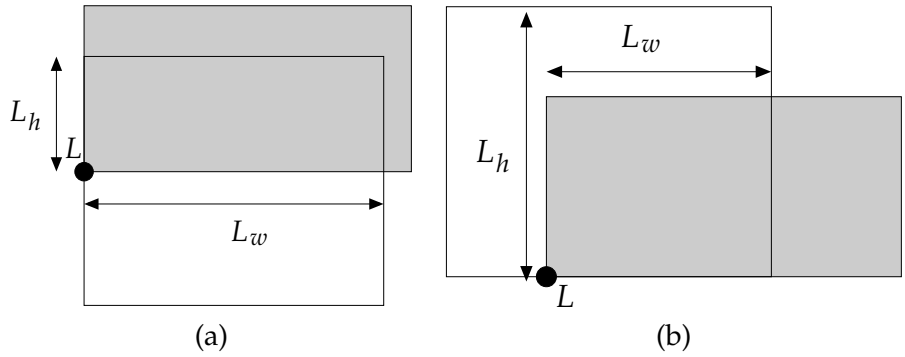


Figure 6.6: *Definition of L_w and L_h for at location L in a pocket. Two different locations of pocket. A module is shown in both (a) and (b) at the location L. L_w is the width of the location space. L_h is the height of the location space. Note that for the lower left location of a pocket L_w is the width of the pocket and L_h is the height of the pocket.*

The guided local search (GLS) approach by Færø et al. [24, 21], introduced a penalty term between each pair of modules. Indeed a similar approach could be used here between a location and module, but the primary problem would be determining when to use a high penalty and when to use a low penalty. In the GLS-framework this was handled by the GLS meta-heuristic by increasing the penalty factor for most overlapping modules at local minima. Færø et al. [24, 21] raised β during the placement algorithm to work first with relaxed placements and then later towards legal placements.

One could probably control location penalties with GLS but it makes less sense since we are not working towards completely legal solutions.

6.2 Neighborhood Reduction

The last of the mentioned problems with the swap-based neighborhood is its size. Fortunately we can determine a region for a module m for which it is guaranteed that the length of nets incident with m will be reduced.

6.2.1 Bounding-Box Net-Functions Revisited

Assume that n is an incident net of m and that all other modules connected to n are static. Also assume that p_l is the leftmost pin connecting m with n and p_r is the rightmost pin connecting m with n. Note that we allow $p_r = p_l$. Let $n \backslash \{m\}$ be the remaining modules connected to n.

Let x_l^n be the x-coordinate of the left-most pin and x_r^n be the x-coordinate of the right-most pin connecting n with modules in $n \setminus \{m\}$. Let $\text{ofs}_x(p_l)$ be the x-offset of p_l and $\text{ofs}_x(p_r)$ be the x-offset of p_r. Now we consider horizontal translation of m.

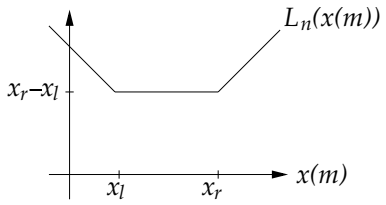


Figure 6.7: *The x-portion of the bounding-box netlength of a net as a function of the x-position $x(m)$ of module m. x_l is the left-most and x_r is the right-most pin of the remaining modules of n. The offset of the pin connecting m with n is zero in this simple example.*

As m is moved right from $x(m) = -\infty$ the left border of the bounding-box of n is moved right and $BB(n)$ decreases until the x-coordinate of p_l reaches the left border of the bounding-box of n ($x(m) + \text{ofs}_x(p_l) > x_l^n$) or p_r is beyond the right border of the bounding-box of n ($x(m) + \text{ofs}_x(p_r) > x_r^n$). If $\text{ofs}_x(p_r) - \text{ofs}_x(p_l) < x_r^n - x_l^n$ it will be the first of these scenarios that occurs.

Now assume this scenario ($x(m) + \text{ofs}_x p_l > x_l^n$). In this case $BB(n)$ remains constant until p_r reaches the right border of the bounding-box. On the other hand if we assume the second scenario ($x(m) + \text{ofs}_x(p_r) > x_r^n$) $BB(n)$ remains constant until p_l reaches the left border of the bounding-box for n. After either of these two last incidents the right border of the bounding-box moves to the right while the left border remains static and $BB(n)$ increases.

By exploiting these observations we can write a piece-wise linear function $L_n(x(m))$ which is the bounding-box netlength as a function of the $x(m)$ since the vertical extend of the bounding-box of n is completely independent of $x(m)$. Assume that $\text{ofs}_x(p_r) - \text{ofs}_x(p_l) \leq x_r^n - x_l^n$ (the first of the two scenarios). Then we have:

$$L_n(x(m)) = \begin{cases} x_r^n - x(m) - \text{ofs}_x(p_l) & \text{for} \quad x(m) \leq x_l^n - \text{ofs}_x(p_l) \\ x_r^n - x_l^n & \text{for} \quad x(m) \in]x_l^n - \text{ofs}_x(p_l), x_r^n - \text{ofs}_x(p_r)] \\ x(m) + \text{ofs}_x(p_r) - x_l^n & \text{for} \quad x(m) > x_r^n - \text{ofs}_x(p_r) \end{cases} \tag{6.3}$$

(A similar version can be written for the second scenario). $L_n(x(m))$ is shown on figure 6.7. Using this observation we may write the x-portion of the bounding-box netlength of the incident nets as:

$$L(x(m)) = \sum_{\{n|m \in n\}} w(n) L_n(x(m)) \tag{6.4}$$

Since $\tilde{L}_n(x(m)) = w(n) \cdot L_n(x(m))$ is continuous piece-wise linear we may write it as:

$$\tilde{L}_n(x(m)) = \begin{cases} a_n^0 \cdot x(m) + b_n^0 & \text{for} \quad x(m) \in I_n^0 \\ a_n^1 \cdot x(m) + b_n^1 & \text{for} \quad x(m) \in I_n^1 \\ a_n^2 \cdot x(m) + b_n^2 & \text{for} \quad x(m) \in I_n^2 \end{cases}, \tag{6.5}$$

with a_n^i, b_n^i, and I_n^i for $i \in \{0,1,2\}$ defined appropriately. With this definition we can write

$$\begin{aligned} L(x(m)) &= \sum_{\{n|m\in n\}} \sum_{x(m)\in I_n^i} \left(a_n^i \cdot x(m) + b_n^i\right) \\ &= a \cdot x(m) + b. \end{aligned} \tag{6.6}$$

For a and b defined for specific $x(m)$. Now consider $x(m)$ moving from $-\infty$ to ∞. Initially $a = \sum_{\{n|m\in n\}} a_n^0$ since $x(m) \in I_n^0$ for sufficiently small $x(m)$. As $x(m)$ leaves an interval I_n^i and it enters the interval I_n^{i+1} a is decreased by a_n^i and increased by a_n^{i+1} since a_n^i leaves the sum and a_n^{i+1} enters the sum. A similar observation holds for b.

This observation allows us to search $L(x(m))$ from left to right by a sweep-line algorithm. Let $BR = \{x \mid x = x_l^n \vee x = x_r^n \text{ for } m \in n\}$ be a set of breakpoints. Set $a = \sum_{\{n|m\in n\}} a_n^0$ and $b = \sum_{\{n|m\in n\}} b_n^0$. Now visit the breakpoints from BR in order from left to right (smallest to largest). As each breakpoint x is visited do the following:

- If x is x_l^n for some net n then subtract a_n^0 and b_n^0 from respectively a and b and add a_n^1 and b_n^1 to respectively a and b.
- If x is x_r^n for some net n then subtract a_n^1 and b_n^1 from respectively a and b and add a_n^2 and b_n^2 to respectively a and b.

Now during this search $L(x(m)) = a \cdot x(m) + b$.

6.2.2 Guaranteed Improving Region

For our purpose the most important element of $\tilde{L}_n(x(m))$ is that regardless of which of the two scenarios for translation of $x(m)$ is true, $\tilde{L}_n(x(m))$ decreases from ∞ on I_n^0, is constant on I_n^1 and increases to ∞ on I_n^2.

Returning to $L(x(m))$ we are now able to prove the following theorem:

Theorem 6.1. *Assume $L(x(m))$ is defined as (6.4) then the following holds:*

1. *$L(x(m))$ is semi-convex (Please see appendix D for our definition of semi-convexity).*
2. *For some x_0 there exists an interval $[x_l, x_r]$ such that $L(x(m)) \leq L(x_0)$ if and only if $x(m) \in [x_l, x_r]$.*

Proof. 1. Each of the functions $\tilde{L}_n(x(m)) = w(n) \cdot L_n(x(m))$ are semi-convex. To see this first note that the netlength goes to ∞ if a module moves either left or right towards $-\infty$ or ∞. Also the three segments of $\tilde{L}_n(x(m))$ have slopes $-w(n), 0, w(n)$ in that order. Therefore $\tilde{L}_n(x(m))$ is semi-convex since it is also continuous and piece-wise linear. Since $L(x(m))$ is a sum of semi-convex functions $L(x(m))$ is also semi-convex according to lemma D.2.

2. This is simply a different formulation of lemma D.4. □

Assume a module m is placed at $(x(m) = x_0, y(m) = y_0)^t$ then theorem 6.1 states that there exists an interval $[x_l, x_r]$ such that if and only if the x-coordinate of m is within this interval the sum of the x-component netlengths of nets connected to m will be less then or equal what they are at $x(m) = x_0$. The interval $[x_l, x_r]$ can easily be calculated by traversing from left to right using the previously defined sweep-line algorithm:

- First calculate $L(x_0)$.
- Use the sweep-line algorithm to move from left to right.
- Determine when $L(x(m)) = L(x_0)$ during this traversal.
- Let the first point for which $L(x(m)) = L(x_0)$ be x_l.
- Let the last point for which $L(x(m)) = L(x_0)$ be x_r.

y-interval Any part of the previous discussing also applies to the y-component of the netlengths of nets connected to m. Therefore a similar interval $[y_d, y_u]$ in the y-direction can determined in the same manner. We omit the details.

Since the x- and y-portions of the bounding-box netlength are independent, the interval for the x-direction and the interval for the y direction can be combined to form a two-dimensional rectangular area in which it is guaranteed that the netlength will be less or equal to the current one. This area is given simply by

$$R = [x_l, x_r] \times [y_d, y_u]. \tag{6.7}$$

It should be noted that there may exist module positions $(x, y)^t \notin R$ which reduce the sum of netlengths. This is true since an increment in x-netlength could be canceled by a similar decrement in y-netlength. However R is easy to calculate and very easy to test against.

6.3 Swap-Based Local Search

Based on the previous sections we can now describe the local search algorithm in more detail. We assume that the pocket-based floor-plan has been constructed prior to the local search and that pockets are inserted into a bin data-structure so that it can easily be determined which pocket overlaps with a position $(x, y)^t$ on the placement area.

To do local search for a module m_0 within this region. the change in augmented objective function is evaluated for positioning m_0 at each location of R. The change is calculated by subtracting old and adding new lengths of nets connected to m_0, and subtracting old and adding new penalty for m_0. If there is a module m at the location

we place it at m_0's original position and evaluate further change in augmented object function. The best of these location is the one with most reduction in augmented objective value. m_0 is placed at this location.

This final version of the local search method is described in algorithm 6.2.

Algorithm 6.2: Swap-based local search

Input(*A placement and module m_0 which should be optimized*) ;
$L_0 = [$ Location of $m_0]$;
$l_{\text{best}} = -\infty$;
$L_{\text{best}} = [\text{none}]$;
$R = [x_l, x_r] \times [y_d, y_u]$ (as defined by (6.7)) ;
foreach *Location L within R* **do**
  Place m_0 at L ;
  if *L is occupied by module m* **then**
    Place m at L_0 ;
  $l = [$ objective value reduction $]$;
  if $l > l_{\text{best}}$ **then**
    $l_{\text{best}} = l'$;
    $L_{\text{best}} = L$;
  Return m_0 and possibly m to their original locations ;
Place m_0 at L_{best} ;
if *module m is at L_{best}* **then**
  move m to L_0
Split the pocket of L_{best} if necessary ;
return *New solution to placement problem*

6.3.1 Orientations

In some VLSI-instances all eight orientations as described in section 2.2.4 are legal. In other cases only a limited number of orientations may be allowed (e.g. mirroring). Finally in some cases only one orientation is allowed.

If more orientations are allowed m_0 is checked for *every* one of the legal orientation at *every* location.

An alternative approach would be to split the change of orientation into a completely separate local search move, so either m_0 would change position *or* orientation but never both. However there is a good chance, that rotating m_0 would increase overlap penalty too much, and therefore such changes in orientation would hardly ever be accepted.

Note that the improving regions contains the current position of m_0 so even if no good swap can be found m_0 is allowed to change orientation.

6.3.2 Fast Evaluation of Netlength Change

In local search it is imperative that the netlength change due to a move or swap can be calculated fast.

If the intervals of the net-function $L(x(m_0))$ are stored as a balanced tree data-structure for each module m, it is possible to calculate the netlength reduction of any position in time $O(\log k)$ when k is the number of nets connected to m_0. Unfortunately whenever a module m_0 is moved it is extremely expensive to update the data-structures since $L(x(m))$ of every other connected module m must be updated. Updating each $L(x(m))$ may require $O(k)$ time since the constants of the linear functions depend on each other and the entire tree must be updated. In total a move would require $O(l \cdot k)$ time where l is the number of modules connected to m and k is the maximum number of nets any of the modules is connected to.

On the other hand most moves of the swap-based algorithm are of the form: move m_0 to $(x, y)^t$, move m to $(x_0, y_0)^t$, move m back to $(x_0, y_0)^t$ and move m_0 back to $(x, y)^t$. In other words m_0 is moved back and forth a number of times. So most of the time we only need to update the data-structure for the second module m. To handle this we can mark if $L(x(m))$ would need recalculation (if m and m_0 were connected). Then when testing m at (x_0, y_0) we could recalculate $L(x(m))$ if needed. Upon return of m_0 the data-structure for $L(x(m))$ would need recalculation once again if the two modules were connected.

Because of the complications involved with the search-tree data-structures we have chosen a slightly simpler method. For each net n we store extreme modules, i.e. the left-most, right-most, upper-most and lower-most modules. Also we store the second-most extreme modules. Now if we wish to calculate the net-length of n without the module m, we can test to see if it is one of the extreme modules. If this is the case we can use the second-most extreme module in that direction to calculate the bounding-box of n without m, otherwise the bounding-box of n is unaffected by the removal of m. The bounding-box of n without m can now be expanded to any new position for m. The concept is illustrated on figure 6.8 and figure 6.9.

Whenever we wish to determine the netlength reduction for a new location of a module m_0, we can calculate the bounding-boxes before m_0 is moved and afterwards in constant time for each net connected to m_0. We also use the on-demand recalculation scheme of the previous section, but this time on the nets. The complete scheme during the inner loop for evaluating the swap of two modules m_0 and m is as follows:

- Calculate the netlength reduction of moving m_0 to its new location by using the extreme and second-most extreme modules.
- Move m_0 to its new location and mark every net connected to m_0 as "dirty".
- Use extreme modules to determine netlength reduction of moving m_0.

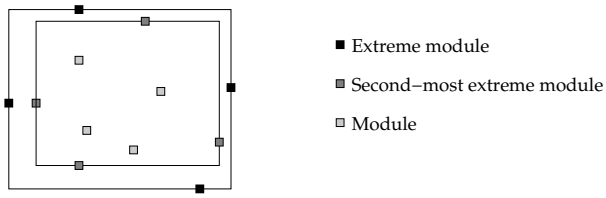


Figure 6.8: *For fast evaluation of bounding-box net-length we store the extreme and second-most extreme modules of each net.*

- For every net connected to m recalculate it if it is "dirty".
- Move m to the original location of m_0 and evaluate netlength change by extreme modules.
- Return m to its original location. The extreme modules of nets are maintained.
- Return m_0 to its original location.
- Recalculate all nets connected to m_0 which are no longer "dirty" and mark all nets connected to m_0 as "non-dirty".

When a module is actually moved at the end of the local search all nets connected to it are always recalculated.

The time needed to recalculate the extreme modules of a net n is $O(|n|)$. The total running time of each step of the inner loop is $O(k_0 + k + l \cdot q)$ where k_0 is the number of nets connected to m_0, k is number of nets connected to m, l is the number of nets m_0 and m share, and q is maximum number of pins connected to any of the l nets.

From section 5 we know that the number of nets a module is connected to on average is about 5. So the number of nets shared by two modules is certainly very small. Finally we also know from 5 that the number of modules connected to each net is about 5 on average. Practical experiments have shown that this scheme is extremely effective compared to calculating netlengths at every location.

Large nets In some of the instances – e.g. avqlarge, avqsmall and pu – we have discovered that some nets contain in the order of thousands of pins. These nets are likely clock-signals or similar. Even with the aforementioned scheme such nets impact performance significantly. Therefore a net is only marked dirty and recalculated if m_0 is moved beyond the inner most extreme modules. Even this simple modification resulted in *great* speed-ups of the local search method.

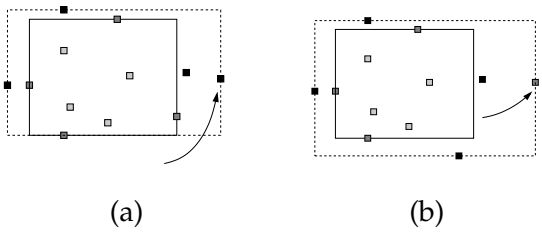


Figure 6.9: *Reevaluation of bounding-box nets based on setup from figure 6.8. (a) One of the extreme modules is moved. The bounding box of the other extreme modules and the second most extreme module in that direction are used to create a new bounding-box. The new bounding-box is expanded with the new location of the moved module. (b) One of the non-extreme modules is moved. The bounding-box of the extreme modules is expanded to include the new location of the moved module.*

7 New Global Placement Heuristic

In this section we will present a new technique for global placement. First we will describe how the quadratic star netlength function is used in conjunction with the legalization algorithm of section 4. Then we will describe how the legalized solution can be used to modify the quadratic formulation and result in a new solution. This we use to create an iterative algorithm.

Quadratic netlength choice We have chosen to use the *star netlength* for quadratic optimization because it generates more sparse matrices than the clique netlength. We have not used a hybrid method which was described at the end of section 2.4.1, but the main difference should only lie in *slightly* longer solution times to the quadratic problem. Note that unlike the clique netlength the star netlength per default is not divided by the number of pins.

7.1 Legalizing Unconstrained Quadratic Placements

By legalizing analytic unconstrained quadratic placements one can generate a solution to the placement problem. However there are a few subtleties to this that we will describe in the following sections.

7.1.1 Unconstrained Quadratic Placement Revisited

As mentioned in section 2.4 the quadratic netlengths can be written in matrix notation as:

$$g(\mathbf{x}) = \mathbf{x}^t \mathbf{C}\mathbf{x} + \mathbf{d}^t \mathbf{x} + f \tag{7.1}$$

Where $\mathbf{x}$ represents the current placement of modules. It was also described how by using the Conjugate Gradient Method it was possible to solve the unconstrained problem:

$$\min_{\mathbf{x}} g(\mathbf{x}) \tag{7.2}$$

By legalizing the solution $\mathbf{x}$ using the legalization algorithm of section 4 we would have a legal solution based on the unconstrained quadratic solution. In general the bounding-box netlength with this initial legal solution is poor. Therefore we initially tried to improve the initial solution by adjusting net contributions.

Linearization Scheme In section 3.1.3 we described a Linearization scheme [79] by Sigl et al. which was used for Gordian-L and also adopted by the force method of Eisenmann and Johannes [19]. The method divides the quadratic star netlength during quadratic optimization with the linear distance between modules and star points of the last solution to create an approximation to the linear star netlength. The new "net-weights" are then used in the quadratic solver to return a new solution. This method is repeated until the sum of differences between linear netlength is less than some ϵ. The method is described in algorithm 7.1. Note however that the linearization scheme is only described for the x-coordinate. The linearization scheme is separated on each coordinate just as the minimization step is. w_0 was set by Sigl et al. to the average width of a module (height for the y-direction).

Algorithm 7.1: Linearization scheme of Sigl et al. [79]

Set iteration number $k = 0$;
foreach *net* $n \in \mathcal{N}$ **do**
  Set current linear star length of n to $g_n^{(k)} = 1$
repeat
  Minimize unconstrained placement with net-weights $\tilde{w}(n) = \frac{w(n)}{g_v^{(k)}}$;
  $k = k + 1$;
  foreach *net* $n \in \mathcal{N}$ **do**
    Recalculate star linear netlength $g_v^{(k)}$ by: ;
    $g_v^{(k)} = \max(w_0, \sum_{p \in N} |\mathbf{A}(p)_x - st_{2x}(n)|)$
until *Current and last linear netlengths obey* $\sum_{n \in \mathcal{N}} |g_n^{(k)} - g_n^{(k-1)}| \leq \epsilon$;

Net-size fraction The linearization scheme did not improve placements significantly. Most likely because modules are not separated during the first iteration. Instead we try to predict the length of nets before the quadratic placement. Nets with many modules will probably be large making the star netlength much larger then the bounding-box netlength. On the other hand small nets will have netlength close to the bounding-box netlength. Therefore we divide the weight of each net with the number of pins and

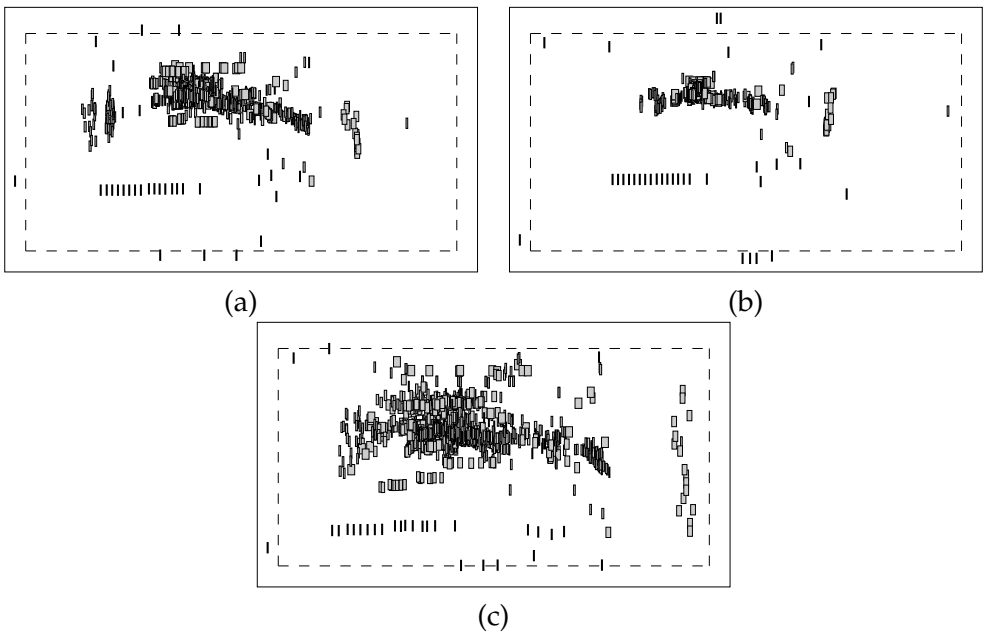


Figure 7.1: *Unconstrained placements of primary2. (a) Standard star netlength placements. (b) Placement with linearization scheme. (c) Placement with net-size fraction.*

set the weight during unconstrained minimization to:

$$\tilde{w}(n) = \frac{w(n)}{|n| - 1}, \quad \text{for } n \in \mathcal{N} \tag{7.3}$$

Note that this fraction has no relation with the fraction explained for the clique netlength in section 2.3.3. The fraction for the clique netlength was to ensure that large nets did not dominate the net-function. Our fraction is introduced to make the star netlength behave more as the bounding-box netlength. Our net-fraction is not related to theorem 2.3 either since this deals with linear star netlength and $RSMT(n)$.

The three unconstrained forms for the primary2 circuit are shown on figure 7.1. The figure illustrates that the net-size fraction method does give better spreading than the other two methods.

7.1.2 Legalizing the Quadratic Placements

The unconstrained placements in general hold a great deal of overlap and the legalization algorithm has difficulty legalizing them without moving modules too much. The setup of the legalization algorithm is as follows:

- Limited placement area and fixed modules are considered, so the legalization will obey all constraints.

- The algorithm uses centered legalization (unless we specify otherwise).
- The algorithm uses the extended envelope sequence-pair placement method.

Also rather than using the complex sequence-pair conversion algorithm we use the simple diagonal-heuristic for determining the sequence-pair of the legalized placement which is slightly faster and seems to work better for massively overlapping placements (see section 9.3.1 for run-time comparisons).

Varied α We have used the extra time gained from the heuristic sequence-pair extractor to search different legalizations by altering α (see section 4.5.2. In general the netlength as function of alpha is roughly a convex curve. this is illustrated for three different circuits in figure 7.2. Although α does seem convex we have decided to simply search an interval for good α values. Therefore we try the legalization algorithm for the following values of α:

$$\alpha = \tan(\frac{i\pi}{40}), \quad \text{for } i \in \{1, \ldots, 20\}. \tag{7.4}$$

We then choose the best solution among the 20. At this point we should note that preliminary attempts at determining a good value for α based on the unconstrained placement, the modules sizes and the size of the placement area failed completely. It was simply too difficult to estimate good values of α.

A simple standard-cell legalization method To test the quality of the sequence-pair legalization we also implemented a very simple *standard-cell* legalization method. The method sorts the modules by y-coordinate and divides them in rows. The number of modules in each row is estimated based on the average modules width. In each row the modules are placed according to x-coordinate.

Figure 7.3 illustrates how the unconstrained placement from primary2 is legalized as an example of the legalization algorithm.

7.1.3 Results for The Initial Placements

In table 7.1 we have shown results for the initial solutions of the unconstrained placements. The table contains four columns; the solution based on the standard unconstrained star netlength, the solution based on a linearized star netlength using the linearization scheme, the solution based on the star netlength with our net-size fraction weights, and the final column gives the results of the simple standard-cell legalization. All results were computed within minutes on a 980 MHz. Pentium III. One can draw a number of conclusions from the table:

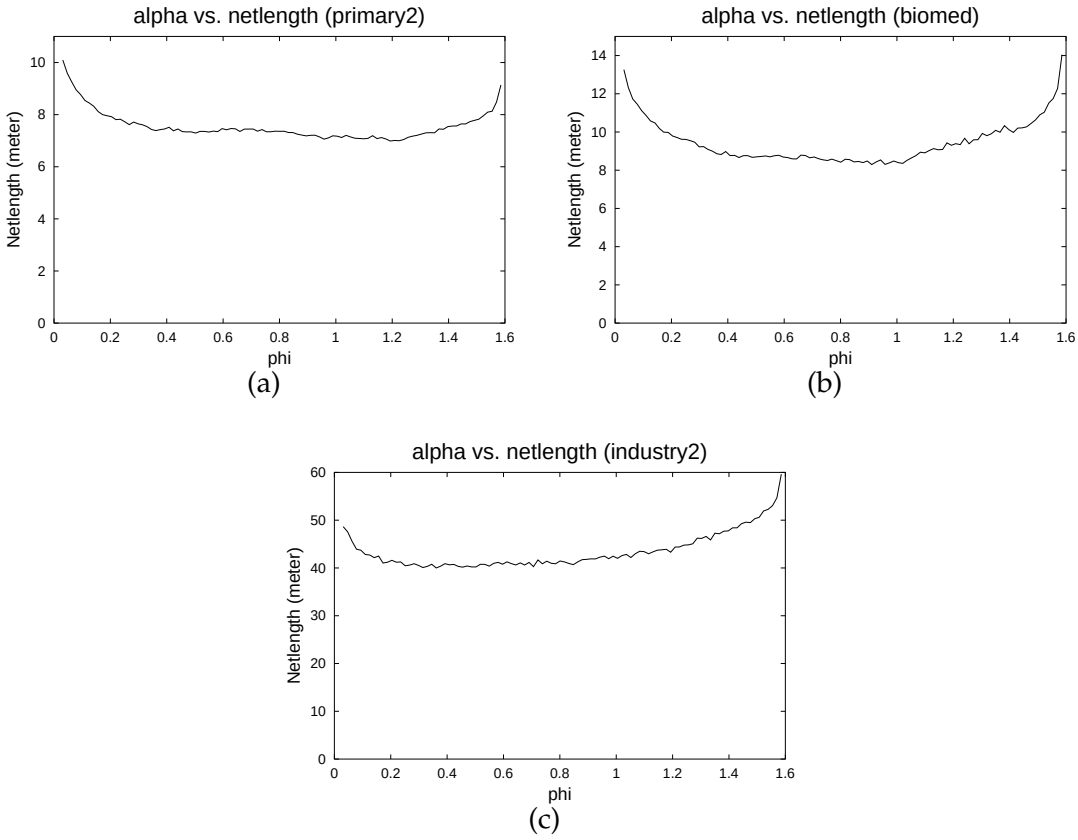


Figure 7.2: *α-values compared with netlength ($\alpha = \tan(\phi)$) for three MCNC standard-cell circuits. The netlength is roughly a convex function of α. (a) primary2. (b) biomed. (c) industry2.*

Circuit	Basic	Net-fraction	Linearization scheme	Standard-cell legalizer
fract	96576	92145	94148	81642
industry1	2687698	2368701	2574084	1717549
industry2	60223530	40132972	44740349	51516819
industry3	133871822	116760980	119017771	151154547
avqlarge	54822592	23404247	28980532	38036796
avqsmall	48491449	21424993	28346937	30653101
biomed	16501145	8305231	13000255	11188979
golem3	433339144	346631291	338263293	318995267
struct	2498590	2098600	2430237	1503554
primary1	1695513	1583839	1787370	1535910
primary2	8421795	6985533	7906193	8627260
decoder	41595421	37557587	39143823	-
pu	241659248	233460664	317062483	-
clk	16688293	14797056	30427228	-
industry2g	80098783	51174857	75153100	-
industry3g	237724726	194539726	257635302	-
ami33K	177236386	155041276	146512641	-

Table 7.1: *Comparison of different net-weight schemes and the resulting legalized placements. The missing numbers (-) arise from the fact that the simple standard-cell legalizer cannot legalize general- and mixed-cell circuits.*

- In general the linearization scheme and net-size fraction scheme work better than the basic setup. Also the net-size fraction is mostly better than the linearization scheme.
- The standard-cell legalizer can outperform the sequence-pair legalizer. This is no surprise since both methods attempt to convert the topology of the overlapping placement to a legal placement. The main difference lies in the underlying methodology. Of course the standard-cell legalizer is not capable of handling general cells.
- Compared to the results of table 3.1 it can be seen that the netlength of the placements in general is two to three times longer than that of published results. This should not come as a surprise either since at this point we have done little to improve the placement.

7.2 Iterative Improvement

Although final placement may be capable of improving the quality it is unlikely that the netlengths can be improved by 50% within reasonable time even by a good final

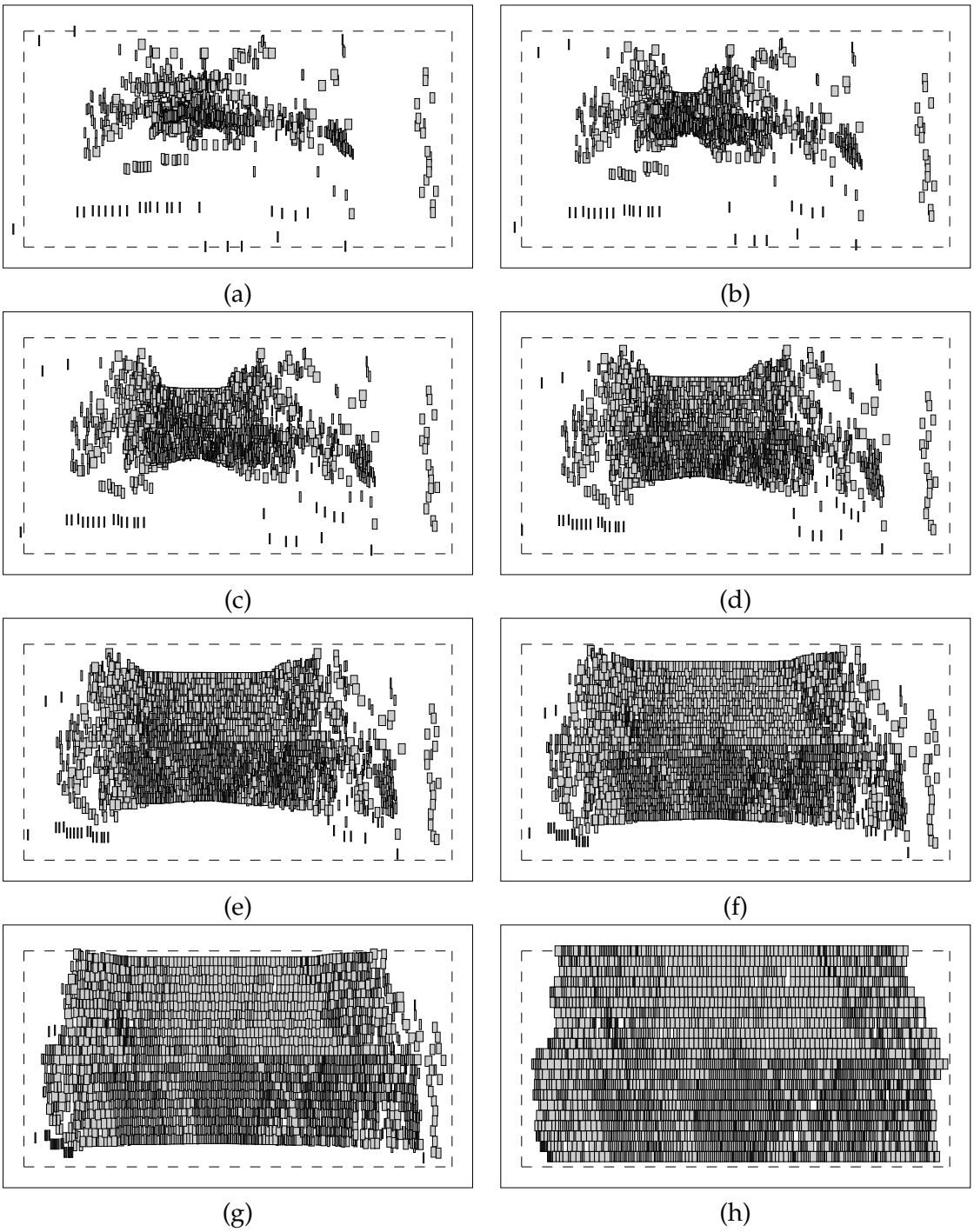


Figure 7.3: *Animation of the legalization algorithm on unconstrained placement of primary2. (a) The optimal unconstrained placement from quadratic star netlength with the net-size fraction scheme. (h) The legalized placement. (b)-(g) animation of modules. The images are interpolated placements between overlapping and legalized positions and are not in any way related to the algorithm. Their sole purpose is to illustrate how modules are moved during legalization.*

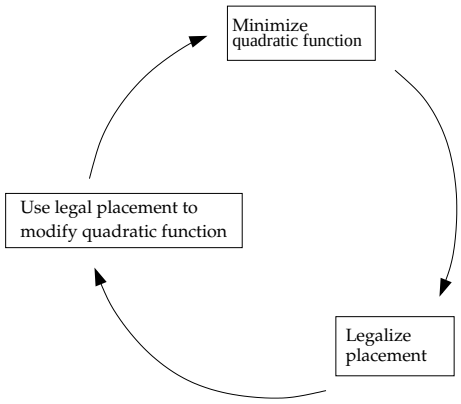


Figure 7.4: *A requirement for the global placer is that the iterative flow use the legalization method in iterative improvement of the quadratic placement by modifying the quadratic objective function.*

placer. Therefore the placements must be improved further to be usable which is the aim of this section.

We have decided to create a new iterative global placement algorithm based on the legalization algorithm. The basic outline of our approach is illustrated on figure 7.4. The legalization algorithm will modify the quadratic formulation. The motivations for using this scheme are:

- The legalization contains no overlap and can therefore "point" in a direction of a good global placement. Also it considers a real placement not just an estimate based on area as the partition based algorithms do. Further unlike the force-based methods which attempt to fill nearby empty regions for each module the legalization algorithm has a more global perspective at where modules can be placed without overlapping.
- If a legalized placement is used to adjust global placement in each iteration then the next solution to the unconstrained quadratic problem could be interpreted as a relaxation of the legal placement.

This will become more clear and we will discuss the two items in more detail when we have presented the algorithm in the following sections. Before presenting the method we will briefly recapitulate how the quadratic function can be modified.

7.2.1 Adjusting the Quadratic Function

To improve the placements we will adjust the quadratic function so that it will be easier to legalize. Altering the quadratic function must be done with care. If the function is altered too much the problem will have little in common with the original quadratic

problem. On the other hand if it is not altered enough the method will have slow convergence. The standard ways to modify the quadratic function are

- **External forces** Add a vector $\mathbf{e}$ to the quadratic function and minimize:

$$g(\mathbf{x}) = \mathbf{x}^t\mathbf{C}\mathbf{x} + \mathbf{d}^t\mathbf{x} + \mathbf{e}^t\mathbf{x} + f \tag{7.5}$$

  by solving

$$2\mathbf{C}\mathbf{x} + \mathbf{d} + \mathbf{e} = 0 \tag{7.6}$$

  This was the method adopted by Eisenmann and Johannes [19]. External forces can drag a module in a specific direction.

- **Artificial nets** Add an artificial net between a module and a static position on the placement area. This was the method adopted by Hu et al. [37]. Unlike external forces the forces from artificial nets are not constant and can be used to drag a module towards a specific position. Specifically a function of the form

$$g(\mathbf{x}) = \mathbf{x}^t(\mathbf{C} + \mathbf{C}')\mathbf{x} + \mathbf{d}^t\mathbf{x} + \mathbf{d}'^t\mathbf{x} + f + f' \tag{7.7}$$

  is to be minimized by solving

$$2(\mathbf{C} + \mathbf{C}')\mathbf{x} + \mathbf{d} + \mathbf{d}' = 0 \tag{7.8}$$

- **Soft constraints** Kleinhaus et al. [49] used a form of soft constraints which were simply part of the objective function.

- **Partition problem** Vygen [86] broke the problem into smaller parts by splitting nets at grid-cell boundaries and iteratively refining grid resolution.

7.2.2 Strategy for Altering the Quadratic Function

The primary problem to be solved is how to use the legalized placement in an iterative flow. The immediate ideas that come to mind are:

1. Use the legalized placement as a hint as to *where* modules should be placed in a non-overlapping placement.
2. Use the legalized placement as a hint as to *which direction* modules should be moved in to remove overlap from the placement.
3. Fix those modules of the legalized placement which are placed at a "good" position.

We have decided to solve this problem by adding artificial nets between well-placed modules in the legalized placement and their position in the legalized placement. The use of artificial nets have the following properties:

1. Modules are dragged towards a specific location not just in the direction of the location which would be the case with external forces.
2. No actual fixing occurs. So even the "good" modules are still allowed movement.

Unfortunately we now have to determine which and how many modules are "good".

7.2.3 Measuring Good Modules

There are a number of ways to measure the quality of a module's location in a placement:

- The distance from the placement in the unconstrained quadratic placement.
- The external forces acting on the module in the current placement. These can be determined by calculating $\mathbf{f} = 2\mathbf{C}\tilde{\mathbf{x}} + \mathbf{d}$, where $\tilde{\mathbf{x}}$ is the legal placement. Each component of $\mathbf{f}$ corresponds to a module. By looking at the component of module m it can be be determined what its external force is. Note that $\mathbf{f} = \mathbf{0}$ for a solution to the unconstrained problem.
- Using the bounding-box netlengths of the incident nets. Areibi et al. [3] presents a utility function which gives low values to modules that are well-placed.

Through preliminary experimentation we have concluded that the distance between the position and the unconstrained quadratic placement is the best way to evaluate a module.

7.2.4 Regions

Unfortunately simply adding artificial nets between the e.g. 1% best placed modules and their legal position gives poor results which will reduce quality of the placement over time. One immediate requirement to the iterative flow is that the modules will be spread increasingly more in each iteration. However the best placed modules – with respect to distance – will be legalized close to the relaxed placement so little spread will arise from this procedure.

To solve this problem we divide the placement area into a number of rectangular regions. Now we add an artificial net between the best placed module in each region and its legal placement.

7.2.5 The Iterative Flow

There is a number of parameters that can be adjusted for the method.

- **Weight of artificial net** Artificial nets should have appropriate weight. This could depend on the quality of the legalized position of the connected module.
- **Number of regions** The number of regions has great effect on the solution. Too many regions will drag many modules towards the legalized placement which may be incorrect. Too few will not alter the quadratic function substantially.
- **Artificial net life time and relaxation** The artificial nets do not need to be static. To allow for new placements the artificial nets could be removed over time or have their weights decreased.
- **Stopping criteria** When to stop the global placement. There are two possibilities; either when overlap is sufficiently small or when a specific number of iterations have been conducted.

Some preliminary experiments have shown that the best setup is as follows:

- Net-weights should not depend on distance to relaxed position. If distance is considered, long artificial nets have either too small or too large influence. Therefore we set weights to some constant ω. A good value for ω will be given during fine-tuning of our global placer in section 9
- Net-weights should be decreased. We multiply them by a constant δ in each iteration. It is important not to remove nets completely since they have perturbed the quadratic function towards its current state and removing the nets will move the quadratic function into an unknown state.
- The number of regions should grow over time. Preliminary experiments showed that multiplying it by 4 in every third iteration until there is an equal number of regions and modules works well. Keeping a number of iterations with an equal number of regions allows the algorithm to undo some of its faults.
- The starting number of regions should be small. Experiments have shown that 64 regions are good. Also we never create more regions than there are modules.
- We let the stopping criteria for global placement be a specific number of iterations which will be revealed in section 9.3.1.

Note that some decisions were made in the previous text. These decisions were made based on many preliminary experiments and seem vital for the algorithm to function properly. With this setup the complete iterative flow is shown in algorithm 7.2

Figure 7.5 illustrates the iterative flow on the primary2 circuit. Figure 7.6 shows the legalization of the last iteration of this procedure.

Algorithm 7.2: Global placement

Input(*Instance of VLSI-placement problem*) ;
Set $k = 64$ regions ;
Set iteration counter $i = 1$;
Set best solution $s_{\text{best}} = [\text{ none}]$;
Let $f(s)$ be the current bounding-box netlength of a solution s;
Set $f(s_{\text{best}}) = \infty$;
repeat
  Solve unconstrained quadratic problem ;
  For a module $m \in \mathcal{M}$ let $\mathbf{p}(m) = (x(m), y(m))^t$ be its center position in the solution of the unconstrained quadratic problem ;
  Legalize the unconstrained problem ;
  For a module $m \in \mathcal{M}$ let $\tilde{\mathbf{p}}(m) = (\tilde{x}(m), \tilde{y}(m))^t$ be its center position in the legal placement ;
  foreach *Artificial net* n **do**
    Set weight $w(n) = \delta \cdot w(n)$;
  Setup $k \times k$ regions ;
  foreach *Region* r **do**
    Determine the module m with legal position in r and smallest distance $d_2(\mathbf{p}(m), \tilde{\mathbf{p}}(m))$;
    Create artificial net n between center of m and $\tilde{\mathbf{p}}(m)$ (weight $w(n) = \omega$) ;
  if $i \equiv 0 \mod 3$ **then**
    $k = \min(k \cdot 4, \sqrt{|\mathcal{M}|})$
  Let s be the current legal solution ;
  if $f(s) < f(s_{\text{best}})$ **then**
    $s_{\text{best}} = s$
  $i = i + 1$
until *Stopping criteria met*;
return s_{best}

Reuse of quadratic solution It should be noted that although minimizing the quadratic function can take a while for the first iteration, using the previous solution as initial solution for the Conjugate Gradient Method will improve speed significantly so subsequent quadratic solutions can be determined in a few seconds even for the largest of the placements.

α-variation During the iterative flow we also test for 20 different legalizations depending on the α-parameter of 4.5.2 as we did for the initial solution (see section 7.1.2).

7.3 Clean-Up Step

The iterative flow of the previous section will give acceptable results. However in general they are still $20-50\%$ worse than previously published solutions (The exact solutions will be revealed in section 9). The main problem is that the legalizer alters the relative order of modules. Therefore a clean-up procedure is applied.

At the end of global placement we will conduct the local search method of section 6 for each module $m \in \mathcal{M}$. When the local search completes we introduce another legalization step to remove overlap from the local search step. This is repeated a number of times. The objective function for the clean-up step is the one of section 6:

$$\min \sum_{n \in \mathcal{N}} w(n) \cdot BB(n) + \gamma \sum_{m \in \mathcal{M}} sizepenalty(L_m, m), \tag{7.9}$$

for some $\gamma \geq 0$.

The outline of the clean-up step procedure is described in algorithm 7.3.

Algorithm 7.3: Clean-up procedure after global placement

Input(*Solution s of the placement problem after global placement*) ;
Let $f(s)$ be the current bounding-box netlength of a solution s;
Set best solution $s_{\text{best}} = s$;
repeat
  Do local search for each module on solution s ;
  Legalize the solution of the local search ;
  Let s be the new legal solution ;
  if $f(s) < f(s_{\text{best}})$ **then**
    $s_{\text{best}} = s$
until *Stopping criteria met*;
return s_{best}

The clean-up procedure has been applied on the global-placement solution of primary2 on figure 7.7.

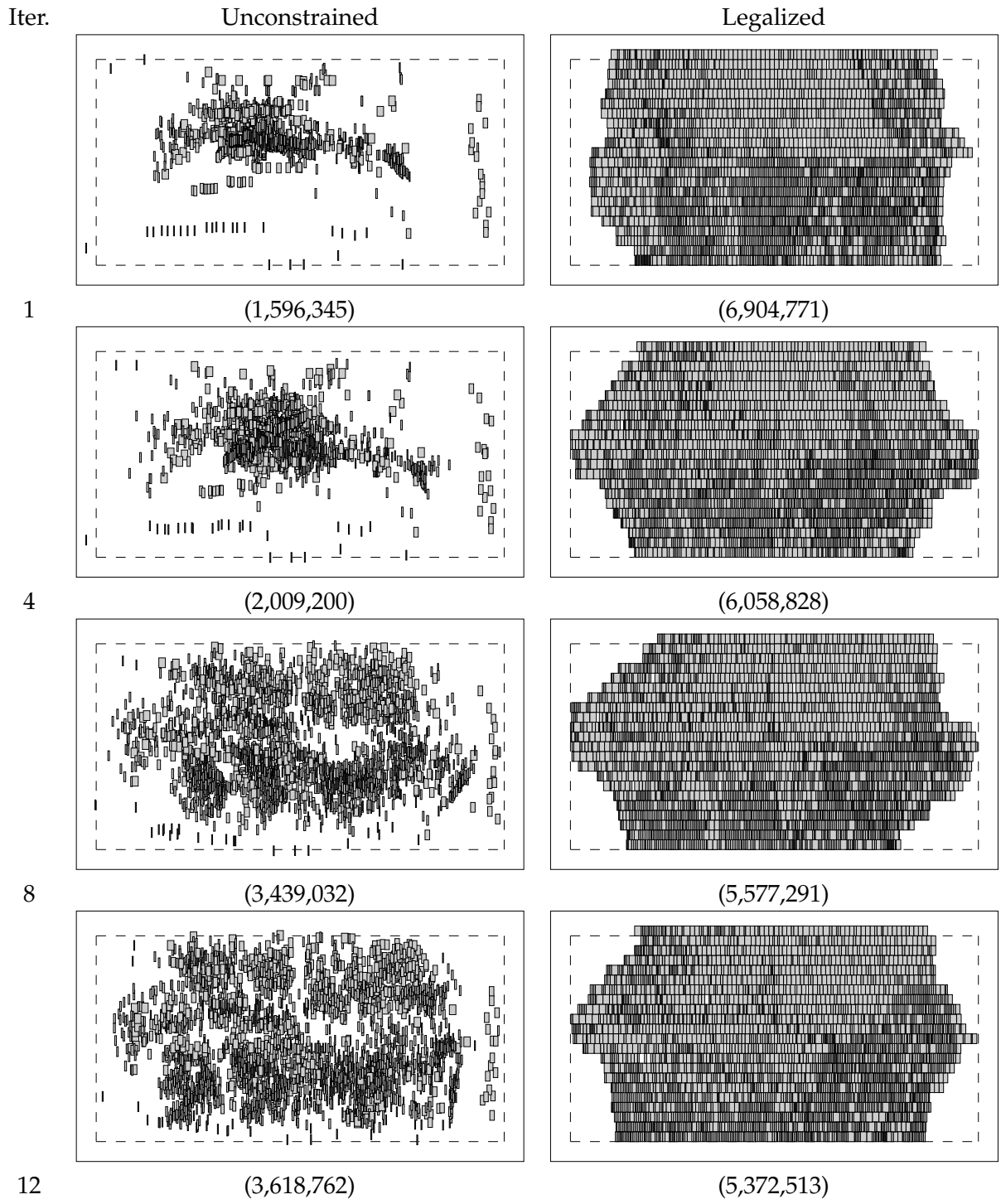


Figure 7.5: Example of global placement on primary2. Images are shown for iteration 1, 4, 8, and 12. The images on the left are the solutions to the unconstrained quadratic problem. The images on the right are legalized solutions. The numbers in brackets indicate netlength in micron at the specific iteration.

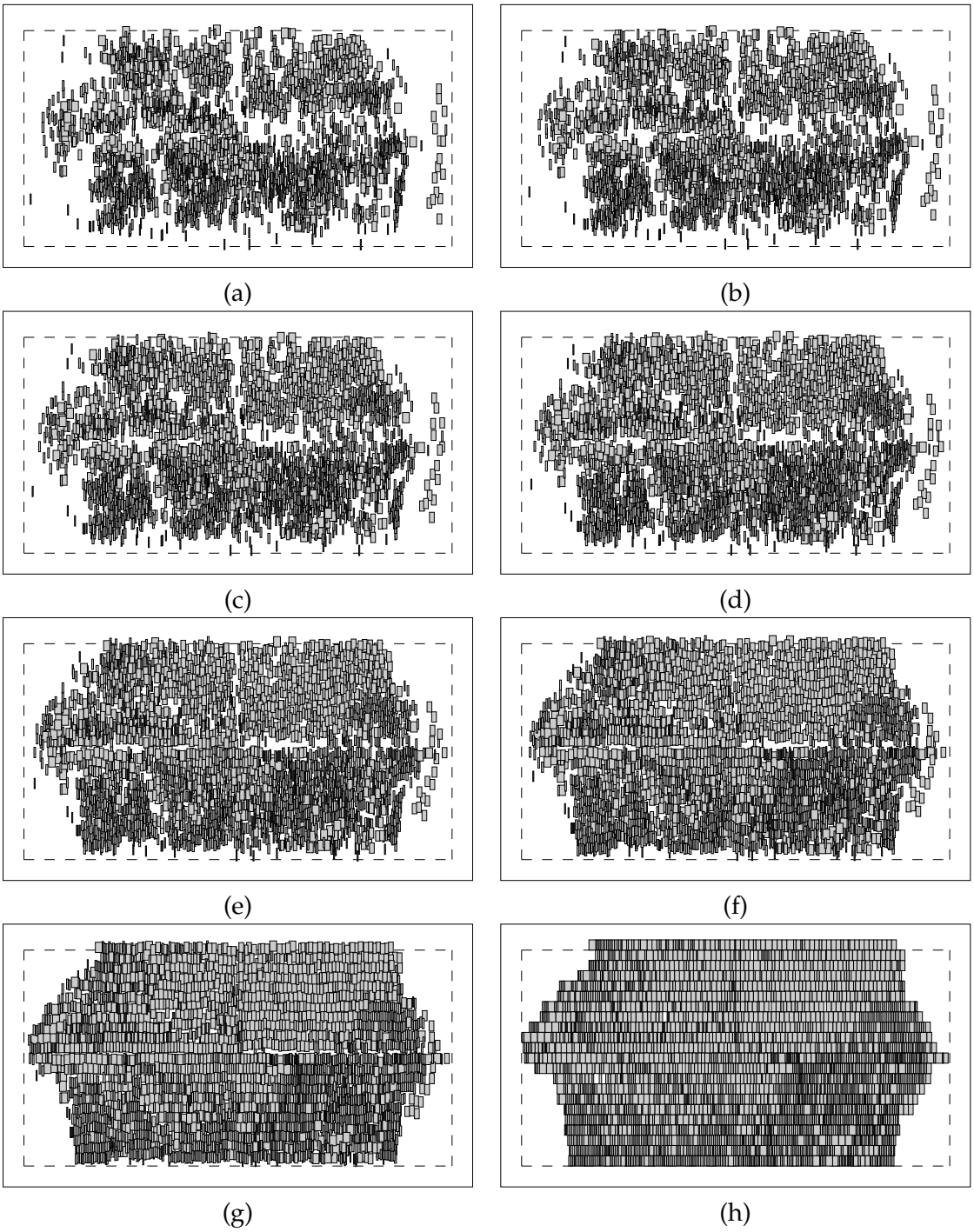


Figure 7.6: *Animation of the legalization algorithm on unconstrained placement of the 12th iteration of global placement of primary2. (a) The optimal unconstrained placement at iteration 12. (h) The legalized placement. (b)-(g) Animation of modules. The images are interpolated placements between overlapping and legalized positions and are not in any way related to the algorithm.*

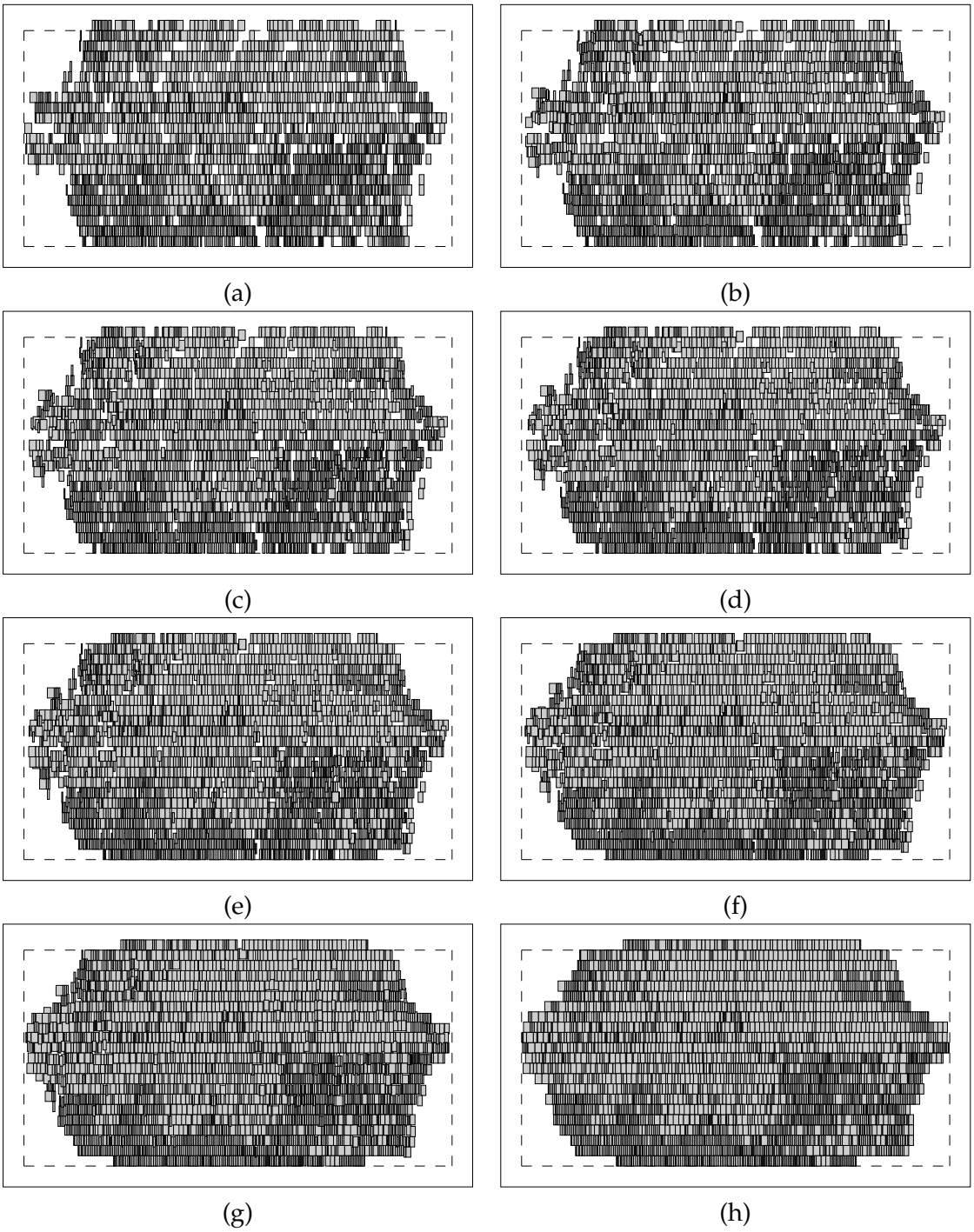


Figure 7.7: *Animation of the clean-up step (a) Solution of primary2 after global placement, legalization and one iteration of the clean-up step. (h) Solution after legalization and clean-up step (netlength: 4,607,676 micron) (b)-(g) Animation of modules. The images are interpolated placements between overlapping and legalized positions and are not in any way related to the algorithm.*

7.4 Connection to Other Methods

Our algorithm is very similar to the ordinary force-based methods (e.g Eisenmann and Johannes [19]). In both cases overlap is attempted removed by spreading the modules. Eisenmann and Johannes use the overlap-induced forces on a bin-structure to calculate external forces. We use the legalized placement. Both techniques have their flaws.

Eisenmann and Johannes technique requires a bin-structure of sufficient resolution to represent the overlap of modules well enough on the placement area. The bin-structure must be used in conjunction with a fast-fourier-transform step to calculate the external forces (see section 3.1.4). If the bin resolution is *very* high this could represent a problem in terms of memory and computational requirements.

Our method does not rely on a bin structure. However the main flaw here is that legalization will tend to push modules in different directions in each iteration. This happens because the legalization algorithm rarely places a module in the same spot in two subsequent iterations. Therefore our algorithm never reaches a state where overlap is completely removed as would be expected from Eisenmann and Johannes algorithm.

Another point is that the legalization algorithm uses a more global perspective to remove overlap while the force-based methods simply use close neighborhood. The effect of this should be faster convergence to good solutions.

The algorithm is also closely connected to the partition based algorithm Gordian of Kleinhaus et al. [49] (see section 3.1.3). Here the modules were constrained to specific regions which were determined by partitioning. We drag well placed modules towards specific regions. The position of each such module is determined by a placement similar to a partition based on area.

Alternative interpretation Instead of interpreting the method as a force-like method it could be interpreted as a relaxation method. In each iteration a legal placement is relaxed to construct a better but illegal placement which is then re-legalized to form a new and hopefully better legal solution than the previous one.

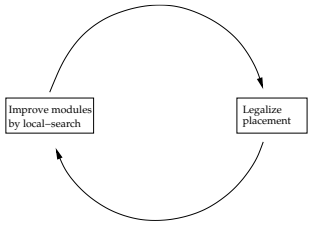


Figure 8.1: *The outline of our final-placement.*

8 New Final Placement Heuristic

In this section we will present a new local search based final placement heuristic. The method uses linear program formulation of bounding-box netlengths and relies on the swap-based local search we described in section 6. The local search of this section also has a larger neighborhood than the local search of section 6. Similar techniques have been used by other authors but our approach *is* a novel technique.

8.1 Outline of Final-placement

Our strategy for final placement is similar to our global placement and is shown on figure 8.1. Using local search we will improve the placement and we will allow for overlapping placements. Then from time to time we will legalize the placement. There are two reasons to legalize the placement during final-placement and not just at the end of final-placement:

- The less overlap a placement has the more accurate the netlength will be with respect to a final legal placement. Therefore to maintain accurate netlength during final-placement the placement should be legalized from time to time. On the other hand legalizing too often will slow final-placement, so the number of legalizations must be balanced. Legalization during final-placement was also briefly discussed in section 3.2.1 when we described TimberWolf 7 [80].
- Legalizing the placement will move many modules and is therefore a way to escape some local minima. On the other hand if modules are moved too much during legalization the improvement steps conducted before the legalization will be lost.

At this point the next question is what form of local search to conduct. The local search method of section 6 would probably work well but it has tendency to move modules into local minima of the form shown of figure 8.2. Here several modules must be moved at one time to improve the current placement.

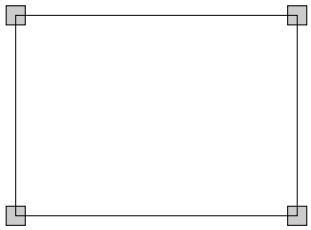


Figure 8.2: *The swap-based local search method of section 6 will create local minima of this type. Each of the four shaded rectangles are modules. The rectangle is the bounding-box of a net. No move of one single module from the net can be cause a reduction in netlength since it would that two modules were moved consecutively..*

To avoid such local minima we have decided to use a stronger but also more demanding local search method. We will extract a collection of connected modules and relax all no-overlap constraints while fixing non-extracted modules. The netlength of the nets connected to extracted modules will then be reduced analytically. This will generally create overlapping placements and therefore we will introduce a local search clean-up step – semi-legalization – based on the local search algorithm of section 6. To guide our new relaxation based local search we will use simulated annealing to escape local minima. The objective function is the same as we used for local search in section 6:

$$\min \sum_{n \in \mathcal{N}} w(n) \cdot BB(n) + \gamma \sum_{m \in \mathcal{M}} sizepenalty(L_m, m), \tag{8.1}$$

for some $\gamma \geq 0$.

The outline of our description of final placement is as follows. First we describe the relaxation based local search method in section 8.2. In section 8.3 we present the simulated annealing heuristic which controls the relaxation based moves. Finally in section 8.4 we summarize the final-placement heuristic.

8.2 Relaxation Based Local Search

The relaxation based method by Hur and Lillis [39] was briefly touched upon in section 3.1.7. They relax sub-circuits (connected groups of modules). The algorithm is for global placement and bin-based. After each relaxation is solved a clean-up step, called node-rippling, ensures that each bin does not have a surplus area of modules. The method was also used earlier by Hur and Lillis [38] during final-placement. Here a "force"-number of each module (similar to the prefer number of section 3.2.1) was used to determine how modules would be legalized. Unfortunately the method as described in [38] relies on equal-sized modules.

We will use as similar approach. The outline of the inner iterations of *our* final-placement heuristic is as follows:

1. **Extract sub-circuit** First we extract a sub-circuits $\mathcal{S}$. A sub-circuit is a collection of connected modules.
2. **Relax sub-circuit** Next we minimize netlength of the sub-circuit while completely relaxing no-overlap constraints of all modules and fixing remaining modules $\mathcal{M} \setminus \mathcal{S}$. This can be done in polynomial time.
3. **Clean-up sub-circuit** We now clean-up the relaxed placement. We do this with a variant of our local search which we described in section 6.
4. **Accept/reject** If the netlength has been reduced or is equal we accept the moves. Otherwise we restore the state of the circuit before extraction and relaxation.

After a number of such iterations we can legalize the placement. The complete final-placement heuristic will repeat this process until some time-criteria or other stopping criteria is met. To control the process we use simulated annealing.

We will explain each of these elements in detail. In section 8.2.1 we explain how sub-circuits may be extracted. In section 8.2.2 we explain how we solve the relaxation of the sub-circuit. In 8.2.3 we explain the clean-up step. In section 8.2.4 we explain how the current state of the circuit along with pockets can be saved. This is necessary since we may reject moves of the relaxation based method which may include many exchanges of module positions. Finally in section 8.2.5 we give the outline of one move in the local search neighborhood for final-placement.

8.2.1 Sub-Circuit Extraction

The first step of each iteration is to extract a sub-circuit $\mathcal{S}$. $\mathcal{S}$ can be extracted in at least three different ways:

- **Randomly - modules based** First a module $m \in \mathcal{M}$ is picked randomly. m is added to $\mathcal{S}$. Now the procedure adds new modules repeatedly to $\mathcal{S}$ by choosing one module m' from $\mathcal{S}$ and adding one module connected to m'. This was the method of Hur and Lillis [38].
- **Randomly - net based** One net $n \in \mathcal{N}$ is picked randomly and its connected modules are added to $\mathcal{S}$. $\mathcal{S}$ is expanded repeatedly by choosing a module m from $\mathcal{S}$ and adding modules of another not previously added net connected to m. Neighbor nets are chosen randomly.
- **Utility-based method** Based on some quality-value of nets and modules which state how well nets and modules are placed, nets or modules may be added greedily but connected nets and modules which have low quality are added first since they are most likely to improve placement. A utility-value for nets and modules were described in [3].

We have chosen a net-oriented scheme rather than a module-oriented scheme because we feel that the relaxation based method works better if modules of an entire net are relaxed. Also ome preliminary tests pointed in the direction that this scheme was better at escaping local minima left by the clean-up step of global placement. We have not tested the utility-based approach but our other attempts with utility based optimization were unsuccessful so we did not pursue it in this context.

Our sub-circuit extraction works as described in algorithm 8.1

Algorithm 8.1: Sub-circuit extraction

Input(*Placement problem and maximum sub-circuit size* k) ;
Set subcircuit $\mathcal{S} = \emptyset$;
Choose net $n \in \mathcal{N}$ randomly ;
Add modules from n to $\mathcal{S}$;
repeat
  Choose $m \in \mathcal{S}$ randomly ;
  Choose n connected to m randomly ;
  Add modules connected to n which have not been previously added to $\mathcal{S}$;
until $|\mathcal{S}| > k$;
return *Sub-circuit* $\mathcal{S}$

8.2.2 Relaxation

The next element to our final-placement heuristic is relaxation of the sub-circuit. Here we will minimize the bounding-box formulation of the nets connected to modules of $\mathcal{S}$ while relaxing the no-overlap constraints.

Let $\mathcal{T}$ be the nets connected to modules from $\mathcal{S}$. To solve the relaxation we construct a linear program formulation of the modules $\mathcal{S}$ and the nets $\mathcal{T}$. Modules connected to nets from $\mathcal{T}$ but not in $\mathcal{S}$ are treated as fixed modules during the relaxation phase.

Using the notation from section 2 the linear program takes the form:

$$
\begin{array}{lrrcll}
\min & \sum_{n\in\mathcal{T}} w(n)(\overline{x_n} - \underline{x_n} + \overline{y_n} - \underline{y_n}) & & & & \\
\text{subject to :} & & & & & \\
 & \overline{x_n} & -x_r(c(p)) & \geq & \text{ofs}_x(p) & n \in \mathcal{T}, p \in n, c(p) \in \mathcal{S} \\
 & & x_r(c(p)) & -\underline{x_n} \geq & -\text{ofs}_x(p) & n \in \mathcal{T}, p \in n, c(p) \in \mathcal{S} \\
 & \overline{y_n} & -y_r(c(p)) & \geq & \text{ofs}_y(p) & n \in \mathcal{T}, p \in n, c(p) \in \mathcal{S} \\
 & & y_r(c(p)) & -\underline{y_n} \geq & -\text{ofs}_y(p) & n \in \mathcal{T}, p \in n, c(p) \in \mathcal{S} \\
 & \overline{x_n} & & \geq & \max_{p\in n}(x(c(p)) + \text{ofs}_x(p)) & n \in \mathcal{T}, c(p) \notin \mathcal{S} \\
 & & & -\underline{x_n} \geq & -\min_{p\in n}(x(c(p)) + \text{ofs}_x(p)) & n \in \mathcal{T}, c(p) \notin \mathcal{S} \\
 & \overline{y_n} & & \geq & \max_{p\in n}(y(c(p)) + \text{ofs}_y(p)) & n \in \mathcal{T}, c(p) \notin \mathcal{S} \\
 & & & -\underline{y_n} \geq & -\min_{p\in n}(y(c(p)) + \text{ofs}_y(p)) & n \in \mathcal{T}, c(p) \notin \mathcal{S}
\end{array}
\tag{8.2}
$$

All variables, $\overline{x_n}, \underline{x_n}, \overline{y_n}, \underline{y_n}$ for $n \in \mathcal{T}$ and $x_r(m), y_r(m)$ for $m \in \mathcal{S}$ are free and represent respectively the boundaries of each net $n \in \mathcal{T}$ and the position of each module $m \in \mathcal{S}$. The last four constraints describe modules not in S (which were static under the relaxation). Notice that the formulation can be split in both an x- and a y-component since the bounding-box netlength is separable in the two directions. For more details on linear programming formulation of the bounding-box netlength see appendix C.2.

We now solve the linear program which minimizes the bounding-box netlength of the nets from T and place modules at the position induced by the solution to the linear program $((x_r(m), y_r(m))^t)$.

Also we set penalties of modules from $\mathcal{S}$ to 0 and update the objective function with new netlengths and penalties. The reason for resetting penalties will become clear in section 8.2.3.

Network-flow interpretation The linear program can be separated in x- and y-component. The dual of either of these linear programs can be interpreted as a minimum-cost network-flow problem with negative edge costs (The details of the flow interpretation are in appendix C.2.2). Each module contributes with one node to the network each net contributes with two nodes and two edges and each pin connecting modules from $\mathcal{S}$ with nets from $\mathcal{T}$ also contributes with two edges. Since the linear program can be written as a network-flow problem we can solve it efficiently in time $O(mn \log(n^2/m))$ [27] where $m = |\mathcal{S}| + 2|\mathcal{T}|$ and $n = 2|\mathcal{T}| + 2p$ when p is the number of pins connecting $\mathcal{S}$ and $\mathcal{T}$.

8.2.3 Semi-Legalization

The relaxation step usually introduces overlap in the placement which must be dealt with. Ideally the legalization algorithm of section 4 could remove overlap but there are two reasons why this is not a good idea:

- Firstly it is computationally expensive to legalize the entire placement after each sub-circuit has been relaxed. Even though the legalization algorithm is fast it still takes a second to legalize a placement containing about 30,000 modules.
- Secondly the overlap induced by the linear relaxation is often severe. As was briefly touched upon in section 3.1.2 linear program relaxation has a tendency to place modules connected to each other in the same spot. Which means that modules are placed on top of each other after the relaxation step of the previous section. However even if the solution to the linear program did not place connected modules directly on top of each other there is a good chance that a relaxed module would be placed on top of some other module on the placement area since placements produced by our legalization algorithm are compact.

Therefore simply using the legalization algorithm at this stage will deteriorate the solution significantly.

Instead we use a variant of the local search method of section 6 to clean-up the placement once again. This procedure we call semi-legalization.

Let $\mathcal{L}'$ be the set of locations occupied by modules from $\mathcal{S}$ before the relaxation step. After relaxation the modules from $\mathcal{S}$ are semi-legalized in some order.

For each module $m_0 \in \mathcal{S}$ the semi-legalization proceeds as follows. Locations in a region R surrounding m_0 are searched in the same manner as in the local search heuristic of section 6. Each location L_0 in R is visited and the change of objective value from placing m_0 at L_0 is evaluated. If a module m is occupying L_0 we attempt to move it to *one* of the locations of $\mathcal{L}'$. Once again we test the objective change of placing m at each location $L \in \mathcal{L}'$. The best combination of location L_0, module m and location L in terms of objective value reduction is chosen and the appropriate movements and pocket manipulations are conducted.

The procedure is described in algorithm 8.2

Algorithm 8.2: Semi-legalization of one module

Input(*A placement problem, a module m_0 which should be semi-legalized, a search-region R, a set of original locations $\mathcal{L}'$*);
$l_{\text{best}} = \infty$;
$L_{\text{best}} = [\text{none}]$;
$L_{0\text{best}} = [\text{none}]$;
foreach *Location L_0 within R* **do**
  Place m_0 at L_0 ;
  if *L_0 is occupied by module m* **then**
    Determine the best location L from $\mathcal{L}'$ for m;
  $l = [$ objective value increase due to moves $]$;
  if $l < l_{\text{best}}$ **then**
    $l_{\text{best}} = l'$;
    $L_{0\text{best}} = L_0$;
    $L_{\text{best}} = L$;
  Return m_0 and possibly m to their original locations ;
Place m_0 at L_{best} ;
if *module m is at $L_{0\text{best}}$* **then**
  move m to L_{best}
Split the pocket of $L_{0\text{best}}$ if necessary ;
return *New solution to placement problem*

There are two parameters for the procedure which we did not explain:

- **Order of modules** In which order should the modules from $\mathcal{S}$ be semi-legalized. This could have an impact on the final solution, since modules placed early will affect steps of later modules and the quality of nets from $\mathcal{T}$.
- **Search region for modules** Only a limited number of exchanged modules $m \in \mathcal{M} \setminus \mathcal{S}$ can be tested for the method to function efficiently.

Order of modules For the order of modules we considered three different approaches:

- Random.
- *Ascending* order of distance between position before and after relaxation step (largest last).
- *Descending* order of distance between position before and after relaxation step (largest first).

Preliminary experimentation showed that the ascending strategy worked best. This makes sense since modules which have moved least were at a good position before relaxation and there is a good chance that they can fall back into their old location without changing the net-boundaries too much from the relaxed placement.

Search region for modules For the search region for modules we tried two approaches:

- **Relaxation region** For each module m with original coordinates $(x(m), y(m))^t$ and new relaxed coordinates $(x_r(m), y_r(m))^t$ let

$$\begin{array}{rclcrcl} x_l^m & = & \min(x(m), x_r(m)), & & x_r^m & = & \max(x(m), x_r(m)), \\ y_l^m & = & \min(y(m), y_r(m)), & & y_u^m & = & \max(y(m), y_r(m)), \end{array} \tag{8.3}$$

  and let m's search region be defined by

$$R_m = [x_l^m, x_r^m] \times [y_l^m, y_u^m]. \tag{8.4}$$

  R_m corresponds to the area between the original location and the new relaxed location.

- **Total relaxation region** Alternatively all modules could have equal search region:

$$R = [\min_{m \in \mathcal{S}}(x_l^m), \max_{m \in \mathcal{S}}(x_r^m)] \times [\min_{m \in \mathcal{S}}(y_l^m), \max_{m \in \mathcal{S}}(y_u^m)] \tag{8.5}$$

Preliminary experiments showed that individual regions worked best of the two strategies. This can be explained by the fact that the region for each module is smaller and can be searched quicker and that the larger search region does not contribute sufficiently to improve the solution. However in practice the region R_m proved too small so we expanded it to double size by subtracting half width and half height of R_m from the lower left corner of R_m and adding half width and half height to upper left corner of R_m.

The argument for the larger search region, R, would be that one of the early modules m could be placed far from the remaining modules expand shared nets. These nets would then be decreased in size if the remaining modules were able to move close to m, which would require a sufficiently large search region.

8.2.4 Store and Restore of Circuit and Pocket State

The circuit state must be stored so that it can be restored if the relaxation based move is rejected. To do this efficiently each move of a module during the relaxation phase is stored along with any modification that occurs with any pockets associated. Before a move of a module is conducted, its current orientation and position are stored in a list of moves. Also if the move results in a pocket-split the data necessary for merging the two resulting pockets are saved in the list of moves.

If the entire relaxation based move is rejected the list of moves is traversed in reverse order and each move is undone one at a time. This way a module can move several times during the semi-legalization phase if e.g. the semi-legalization of a later module of $\mathcal{S}$ moves an earlier module of $\mathcal{S}$.

8.2.5 Final-Placement Move

We are now ready to describe the complete neighborhood move of the final-placement heuristic.

1. A net n is extracted at random.
2. n is used to construct a set of modules $\mathcal{S}$ which are connected and constitute a sub-circuit $\mathcal{S}$
3. The current value of the objective function is stored. The state of pockets and locations of modules $\mathcal{S}$ are also stored.
4. The sub-circuit of $\mathcal{S}$ is relaxed using the linear-program formulation of the bounding-box netlength without no-overlap constraints. Penalties for modules of $\mathcal{S}$ are set to 0. Netlengths for nets incident with modules from $\mathcal{S}$ are recalculated to match the relaxed placement. The value of the objective-function is also recalculated to match the new state.

5. The resulting placement is now semi-legalized. Modules from $\mathcal{S}$ are semi-legalized in order of least moved first. The semi-legalization process searches for a candidate position for each module and possibly a move of a module from a location to one of the original locations of the modules from $\mathcal{S}$ such that the increase of objective value is least.

6. The value of the objective function before relaxation and after semi-legalization are now compared. If the *entire* relaxation based move resulted in an overall improvement or zero-change of objective-value the entire list of moves conducted during semi-legalization is accepted. Otherwise all moves are rejected. Note that the change in objective function can be calculated during semi-legalization and does not require the otherwise very computationally intensive evaluation of the objective function.

7. If the relaxation based move is rejected, moves conducted during semi-legalization are undone one at a time to restore the original state of the circuit and pockets.

The entire procedure is illustrated on figure 8.3

Comments on accept We accept zero moves. The argument for this is that two connected modules may exchange places without reduction of netlength by the procedure described above. This increases the possibility of escaping local minima.

8.3 Simulated Annealing for Controlling Moves

To control the flow of the relaxation based local search we use the simulated annealing (SA) meta-heuristic. SA was chosen because experiments with the guided local search (GLS) heuristic failed and no appropriate alternative for meta-heuristic could be found. This section is divided in two parts. First we give a brief introduction to simulated annealing. Then we explain how we use it in conjunction with the relaxation based local search of the previous section.

8.3.1 Brief Introduction to Simulated Annealing

SA was first used for combinatorial optimization by Kirkpatrick et al. [48] and has been used previously for VLSI-placement optimization as described in sections 3.1.5 and 3.2.1. Simulated annealing exists in many variants. We use a modified version of a simulated annealing which was described by K. Dowsland [17]. Our variant of Dowsland's version is described in algorithm 8.3

The algorithm deserves a few comments. In each iteration a solution is picked randomly. If the solution is better than the previous solution we accept it and continue

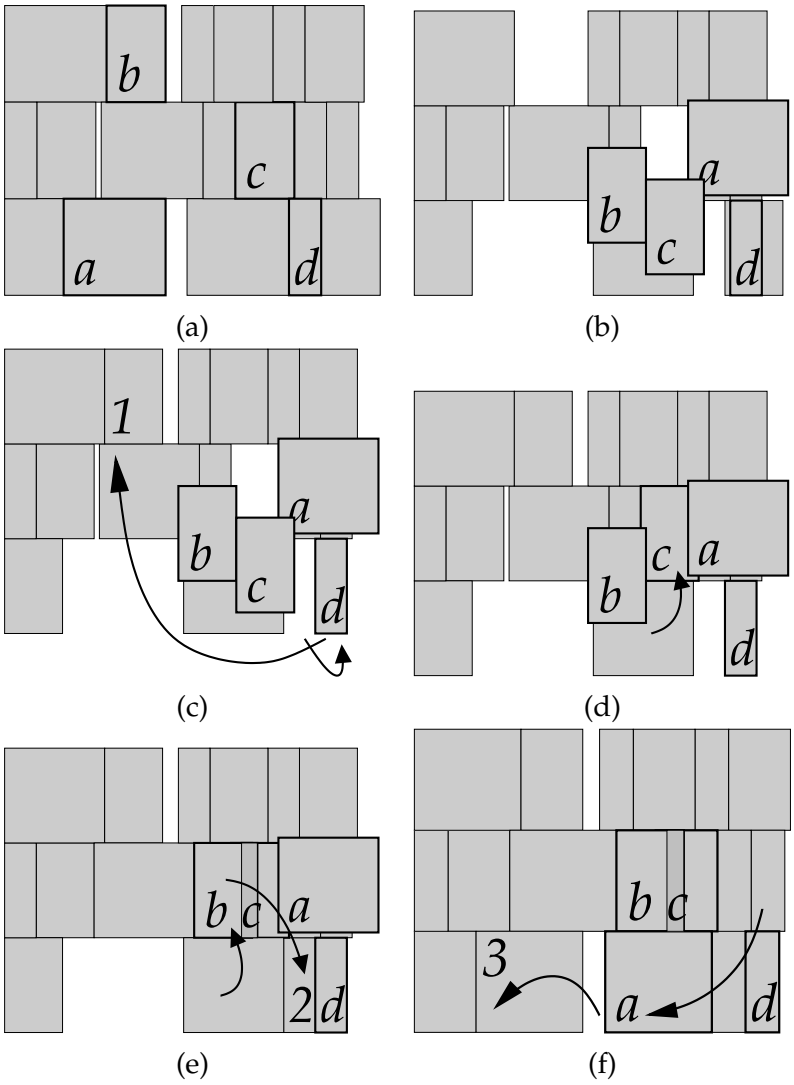


Figure 8.3: *Outline of a relaxed local search move. (a) A sub-circuit $\mathcal{S}$ is extracted consisting of modules $\mathcal{S} = \{a, b, c, d\}$. (b) The sub-circuit is relaxed. Note that a, b, c and d may not end up in the same place since nets connecting individual modules may differ. (c) Modules are semi-legalized in order of least movement during relaxation. d is first and semi-legalization results in movement of module 1 to the original location of b. (d) Next module is c which fall into its original place. (e) Now b is semi-legalized which results in movement of a module 2 to the original location of d. Placing b results in overlap which is penalized in the objective function. (f) Finally a is placed at 3's place and 3 is positioned at a's original place.*

Algorithm 8.3: Variant of Simulated Annealing by Dowsland [17]

Let $f(s)$ be value of objective function for a solution s ;
Select an initial solution s_0 ;
Select and initial temperature $t_0 > 0$;
Select a temperature reduction parameter $K \in]0, 1[$;
repeat
  Select $s \in \mathcal{N}(s_0)$ at random ;
  $\delta = f(s) - f(s_0)$;
  if $\delta < 0$ **then**
    $s_0 = s$;
  else
    Select random $x \in]0, 1[$;
    if $x < e^{\frac{-\delta}{t}}$ **then**
      $s_0 = s$
  if *Solution s was accepted* **then**
    $t = K \cdot t$
  if *Reset criteria is met* **then**
    Set $t = t_0$;
until *Stop criteria met*;
return s_0 *as solution to problem*;

with the new solution. If it is not better than the previous solution we may still accept it if, for some random number x, $x < e^{\frac{-\delta}{t}}$ holds. The value t is the current "temperature". t is reduced by a factor K whenever we have accepted a solution. To better enable the search to escape local minimum we have included a reset strategy which, whenever some reset-criteria is met, resets the temperature of the system. This version has only three variables s_0, K and t_0. Reset- and stop criteria will be specified shortly.

8.3.2 Simulated Annealing for Relaxation Based Local Search

It is straight forward to let simulated annealing control the relaxation based local search. The neighbor solution s is given by the relaxation based local search procedure which starts with a random net and adds modules to the net at random. If the solution is not accepted we simply restore the state of the previous solution. The outline is given in algorithm 8.4.

The cooling schedule based on acceptances was chosen because we believe it will focus search towards local minima. However we hope that the reset method will allow the search to escape local minima from time to time. As will be clear from the following section the initial solution s_0 will generally already be good so the simulated annealer will only work from good initial solutions. Therefore we expect that resetting the *fine-tuned* annealer will not move the search *too* far away from the current solution. The

Algorithm 8.4: Simulated Annealing for relaxation based local search

Let $f(s)$ be the value of objective function for solution s ;
Select an initial solution s_0 ;
Select and initial temperature $t_0 > 0$;
Select a temperature reduction parameter $K \in]0, 1[$;
repeat
  Select net $n \in \mathcal{N}$ at random ;
  Solve relaxation based local search of n ;
  Let s be solution after relaxation based local search ;
  $\delta = f(s) - f(s_0)$;
  if $\delta < 0$ **then**
    $s_0 = s$
  else
    Select random $x \in]0, 1[$;
    if $x < e^{\frac{-\delta}{t}}$ **then**
      $s_0 = s$;
  if *Solution s was accepted* **then**
    $t = K \cdot t$
  else
    Restore to solution s_0 ;
  if *Reset criteria is met* **then**
    Set $t = t_0$;
until *Stop criteria met*;
return s_0 *as solution to VLSI-placement* ;

exact reset-criteria will be given in section 8.4.2

Comments on choice of simulated annealer Simulated annealing is almost a science field by itself. Our choice of simple simulated annealer is merely guesswork as to what will work well. A complete and thorough testing of different cooling strategies and other modifications would be beyond the scope of this thesis and we leave it to future work.

8.4 Complete Final-Placement Outline

We now describe the complete final-placement heuristic. The final-placement heuristic does combinatorial search on an initial solution returned by global placement. The combinatorial search is controlled by the simulated annealing for relaxation based search. Whenever some stopping criteria is met the combinatorial search is halted and the current possibly overlapping solution is legalized using our legalization algorithm. When the solution has been legalized the combinatorial search continues from the new solution.

The outline of the entire final-placement is given in algorithm 8.5.

Note that we accept the legalized solution s always. This is to allow the simulated annealing to continue from where it left off before legalization.

One important element is missing from the final-placement outline; the legalization criteria.

8.4.1 When to Legalize

It is not easy to determine when the solution should be legalized. Different criteria comes to mind:

- **Overlap limit** Legalize when the amount of overlap in the placement increases beyond some upper limit. This way overlap can be controlled however good solutions may be missed because they simply do not carry enough overlap for them to be legalized. When the overlap increases beyond the limit the legalization algorithm may also have difficulty with removing overlap and the otherwise good solution before legalization would be completely lost.
- **Time** Legalize after a specific amount of time spend on local search or a number of local search iterations have passed. This way we can control the amount of overlap in the placement and ensure that legalized solutions are generated frequently. However time may not easily capture the underlying moves of the combinatorial search. Maybe the combinatorial search is fighting to escape local

Algorithm 8.5: Final-placement using SA and relaxation based local search

Let $f(s)$ be the value of objective function for solution s ;
Let s_0 be solution from global placement ;
Let s_{best} be the current best solution ;
Select and initial temperature $t_0 > 0$;
Select a temperature reduction parameter $K \in]0, 1[$;
repeat
  repeat
    Store current state of circuit ;
    Select net $n \in \mathcal{N}$ at random ;
    Extract sub-circuit $\mathcal{S}$ based on n of maximum size k at random ;
    Relax $\mathcal{S}$ using linear program formulation ;
    Semi-legalize $\mathcal{S}$;
    Let s be solution after semi-legalization ;
    $\delta = f(s) - f(s_0)$;
    if $\delta < 0$ **then**
      $s_0 = s$
    else
      Select random $x \in]0, 1[$;
      if $x < e^{\frac{-\delta}{t}}$ **then**
        $s_0 = s$;
    if *Solution s was accepted* **then**
      $t = K \cdot t$
    else
      Restore circuit to solution s_0 ;
    if *Reset criteria is met* **then**
      Set $t = t_0$;
  until *legalization criteria met*;
  Let s be solution after sequence-pair legalization of s_0 ;
  Set $s_0 = s$;
  if $f(s_0) < f(s_{\text{best}})$ **then**
    $s_{\text{best}} = s_0$;
until $Time > MaxTime$;
Return s_0 as solution to VLSI-placement ;

minima or is moving steadily towards a local-minima when we interrupt it. The legalization moves modules so some of this work would be lost.

- **Legalize after fraction modules moved** Legalize after a fraction of the modules from $\mathcal{M}$ has been moved by the relaxation based local search method. This way we only legalize when a specified amount of change has been recorded in the solution and it can still control overlap to some degree. Another reason to use this model is that it is likely that a certain amount of improvement of the objective value is needed to account for the possible loss due to movement of modules during legalization.

We have chosen the last of these strategies – Legalize after fraction of modules moved – since it is independent of problem size and it allows us to control overlap to some degree without depending on the overlap to increase beyond some limit. Preliminary testing also showed that this method was superior to the time or iterations count method. The iterations count method had a tendency to legalize too rarely in the early stages when there are plenty of good moves and too often in the later stages when only few relaxation moves are accepted. Thus the legalization criteria is:

> **Legalization criteria:** When $q \cdot |\mathcal{M}|$ modules have moved as a result of relaxation based local search ($q > 0$).

8.4.2 Variables for Final-Placement

The final placement has several variables which can adjusted and may influence the quality of final-placement. These are:

- **Overlap penalty** γ The overlap penalty determines how much overlap is allowed in local search solutions.
- **Fraction of modules moved** q The amount of modules to move before legalization.
- **SA parameter – initial temperature** t_0 The initial temperature of the simulated annealer.
- **SA parameter – cooling value** The cooling parameter. The temperature t drops by $t = K \cdot t$ whenever a move is accepted.
- **Sub-circuit size** k The size of the sub-circuits extracted by relaxation based local search.

With the exception of the sub-circuit size these parameters will be fine-tuned with experiments in section 9. Large sub-circuits are more difficult to semi-legalize and slows the heuristic. Small sub-circuits weakens the heuristic.

sub-circuit size: For the sub-circuit size we have determined through preliminary experiments that a value of 10 is good.

Reset criteria The reset criteria has not been discussed previously. We have chosen the simplest; reset after r *legalization* iterations. The exact value of r will be determined in section 9. The argument for not resetting after a number of *relaxed local search* iterations is that this way the legalization may miss the local minimum reached before reset.

8.5 Unsuccessful related approaches

We tried many other approaches for final-placement all of which were unsuccessful. Some of the approaches were related to the previously described approach.

- **Extracting cycles** After relaxation of the sub-circuit $\mathcal{S}$ we only allowed modules at an original location of another module from $\mathcal{S}$. In general this is insufficient. Therefore we tried to extract "cycles" of moves. If $\mathcal{S} = \{a, b, c, d, e\}$ and a was positioned at b's old position, b at c's old position, c at a's old position, d at e's old position and e at d's we would have two cycles of moves. If the entire new placement had more netlength it would be rejected. Instead each of the cycles would be evaluated and accepted if they individually reduced netlength.

- **k-moves** Relaxation based local search has a very large neighborhood. Another large neighborhood would be to consider k-moves. Instead of considering simple swaps of pairs of modules we would consider creating cycles of a maximum of k modules. The details of this method are in section A.3.

- **Guided local search I** Guided local search based final placement as implemented by Færø et al. [24, 21] included overlap penalties and connection penalties between connected modules. We tried a similar approach but in a swap based neighborhood and with overlap penalty based on our γ-scheme. The GLS-scheme penalizes pairs of overlapping modules to move towards legal placement. This strategy made less sense in our context so the overlap penalties were dropped. The connected penalties however proved insufficient to improve the placement. Even combined with relaxation based local search.

- **Guided local search II** We also tried a completely different version based on the guided local search meta-heuristic. Here features were nets which violated some lower bound on net-size ($2\sqrt{[\text{area of connected modules}]}$). This lower bound was introduced in [3] to describe utility of modules. However we discovered that good solutions can have large nets which violates the lower bound and that good placements in general are not directly related to the size of the connected nets.

One of the main problems with the strategy may also be that there is no easy way to remove a feature for a net. Even relaxation does not guarantee to remove such features.

9 Experimental Results

In this section we will describe our experimental results. First we describe our implementation which is called G/Flegal (G is for global and F is for final). Then we conduct experiments on the global placement heuristic. These will include fine-tuning, benchmark results and finally comparison with an implementation of another global placement heuristic. Concluding this we will conduct experiments with final placement. As for the global placement heuristic we will fine-tune the final placement heuristic and present final benchmark results.

Netlength unit In the following section netlength is reported in microns unless stated otherwise.

9.1 Implementation

We have implemented the sequence-pair, local search, global and final placement heuristics of the previous sections in C++.

Standard Template Library (STL) was used for most data-structures. For random number generation the C-function `rand()` was used.

The QT-package by Trolltech[10] was used to create a graphical user interface, which was a great help to visualize the heuristics (see figure 9.1).

For solving the unconstrained quadratic minimization problem we implemented both diagonal- and cholesky-precondition versions of the Conjugate Gradient Method. However we had difficulties with numerical stability in relation to cholesky-preconditioning and therefore the following results were achieved by diagonal-preconditioning.

The linear program formulation of relaxation based local search is solved by CPlex and as a standard linear program not as a network-flow problem. Our implementation also include source code to link to the freeware GLPK-solver library[11].

Eisenmann and Johannes [19] force-based global placement was also implemented. Initially to increase our understanding of the force-based methods but in this section we will use the implementation to compare with our method.

A user manual for the program is provided in appendix E.

9.2 Benchmark System

All tests were conducted on a number of identical machines. The machine configurations were:

[10] www.trolltech.com

[11] GNU Linear Programming Kit (http://www.gnu.org/software/glpk/glpk.html)

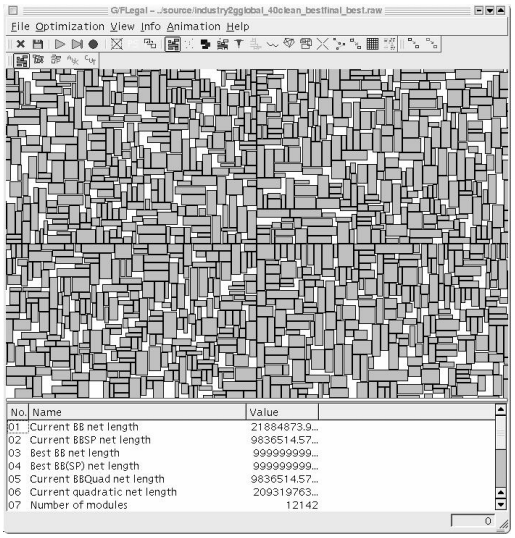


Figure 9.1: *Screen-grab for our placement tool. The graphical user interface allowed us to visualize the underlying algorithms and the placement methods.*

- Dual Pentium III at 930 MHz. No advantage was taken of dual processing capabilities.
- 1 Gb ram.
- Standard installation of Linux Redhat 7.3.
- G/Flegal was compiled with gcc 3.1 and the `-O3` optimization parameter.

9.2.1 Benchmark Circuits

Our heuristic is geared towards placement of general-cell circuits but we will also do placement on standard-cell circuits as a reference to the quality of the general-cell placements and to compare the placement heuristics with those of other authors.

We will therefore use our placement heuristic on all circuits described in section 5 except the small MCNC-benchmarks. Our global and final placement heuristics are simply not geared for such small circuits, and we deem that best netlength would be better reached by methods specifically designed for these circuits. The circuits we choose not to use are the macro-cell circuits (apte, xerox, hp, ami33, ami45) and the smallest standard-cell circuit (fract). We will conduct tests on all other circuits from section 5 including the new benchmark circuits.

Spacing and orientations Except from the ami33k circuit the design rules of all circuits require spacing between modules and this is accounted for by expanding mod-

ules (see section 4.4.2).

For the new general cell circuits primary2g, industry2g, industry3g and ami33k we allow all eight orientations during optimization. Modules of the remainder of the circuits are only allowed one orientation.

9.3 Experiments for Global Placement

First we fine-tune some of the parameters of global placement and the clean-up step. Based on these adjustments we will give results of global placements. This will be followed by comparison of our heuristic for global placement with an implementation of the force-based heuristic of Eisenmann and Johannes (see section 3.1.4).

9.3.1 Fine-Tuning Quadratic Modification

In this section we fine-tune the parameters of the global placement step. The parameters are:

- Number of regions in each iteration.
- Legalization algorithm.
- Initial artificial net-weight ω.
- Fall-off value of artificial net-weight δ.
- Number of iterations for global placement.
- Overlap penalty γ for clean-up step.
- Number of iterations for clean-up step.

Through preliminary tests we concluded that the number of regions at any iteration should be $2^k \times 2^k$ for $k = \lfloor(\text{iteration number} + 9)/3\rfloor$, however we will tune the other parameters. It should be noted that the parameters are likely dependent on each other. This will complicate testing a great deal. Therefore we will idealize our assumptions and assume that parameters are to some extend independent.

Another issue is which circuits to use to determine good values of these parameters. The circuits are similar in some respects and different in others. This will complicate testing. For fine-tuning the global placement heuristic we will generally use a subset of circuits that we expect to represent all circuits well.

	Placement Algorithm Comparison		
Circuit	LCS	Standard envelope	Extended Envelope
industry2	53,640,136/158	29,035,808/145	25,777,463/148
industry2g	62,138,408/309	45,434,113/256	30,602,647/256

Table 9.1: *Comparison of sequence-pair to placement setup. The column of LCS is placements based on the original sequence-pair formulations but implemented with the weighted longest common subsequence method. Standard envelope is semi-normalized placement with standard-envelope. Extended envelope is semi-normalized placement with extended-envelope. Numbers are 'netlength'/'run time in seconds'. Please note that the implementation of the LCS algorithm may be inefficient.*

Preliminary legalization setup Apart from industry1 all circuits are legalized using *complex* centered sequence-pair legalization as described in section 4.5.1 during global placement. We have discovered that industry1 benefits significantly from a legalization with the lower-left corner of the placement area as origin for the sequence-pair algorithm. So for the industry1 circuit we use lower-left corner of the placement area and only one region for legalization.

Legalization Algorithm

We begin by presenting some insight to the legalization algorithm for global placement. We first illustrate the necessity of the extended envelope algorithm. For this purpose we do 20 iterations of the global placement algorithm on the two circuits (industry2 and industry2g). Note that industry2 is a standard-cell circuit. To convert the placement to a sequence-pair we use the heuristic conversion (algorithm 4.1) for this test. We compare the placement results after the 20 iterations of the three methods: ordinary sequence-pair placement (based on the weighted longest common subsequence method), semi-normalized sequence-pair placement and finally our extended semi-normalized placement. The netlength-results and run times are reported in table 9.1 and represent general results.

From the table it should be clear that the extended envelope based method is the better of the three. Figure 9.2 is the legalized placement of the 20th iteration of industry2g. As can be seen the standard sequence-pair method is completely incapable of packing the general-cells of industry2g. The standard envelope method can do this to some degree but the packing violates the placement area severely. Finally our extended method seem to produce an (almost) legal placement.

Preliminary tests have shown that placement-to-sequence-pair heuristic (algorithm 4.1) is faster than the algorithm (algorithm 4.2) while producing results that are almost equal. To give the reader some sense as to the difference of the two algorithms we present results for two different circuits (biomed and industry2) and 20 iterations of the global placement algorithm (untuned) in table 9.2. As can be seen from the table

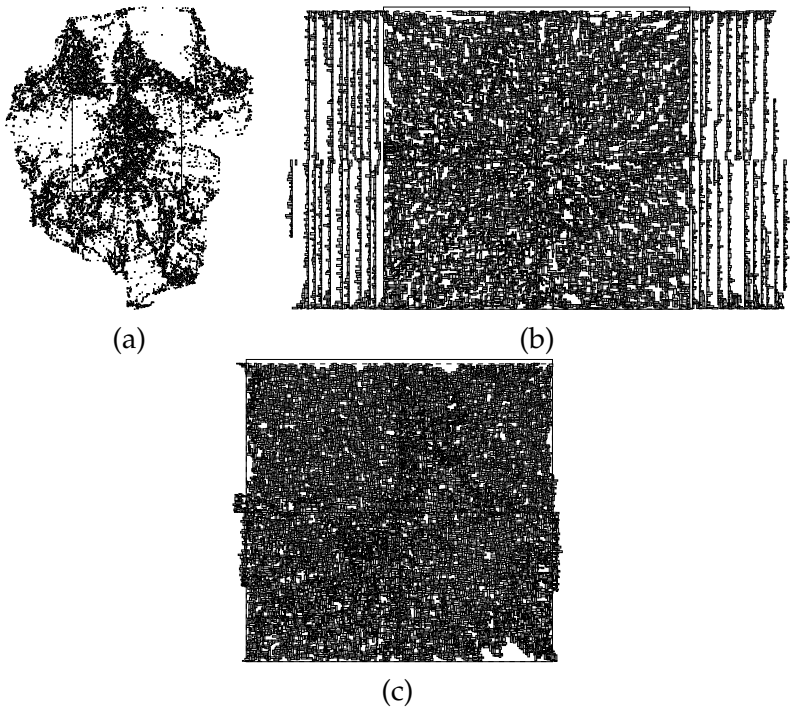


Figure 9.2: *Comparison of the three placement algorithms. This is the legal placement of the 20th iteration of untuned global placement of industry2g. (a) Standard sequence-pair placement. (b) Semi-normalized sequence-pair placement. (c) Extended semi-normalized sequence-pair placement. Please note that all three placements violate the placement area which is the large unfilled rectangle in the middle. Note that the standard sequence-pair algorithm has placed modules very uncompacted. The reason for the columnized placement in (b) arises from the placement algorithm's attempts to maintain the placement within the placement area.*

	Placement to Sequence-pair comparison			
Circuit	Heuristic SP Netlength	Run time seconds	Algorithm SP Netlength	Run time seconds
biomed	6,043,345	78.5	5,914,488	160
industry2	25,448,553	163	24,849,633	364

Table 9.2: *Comparison of placement-to-sequence-pair algorithm during global placement. Heuristic SP is the heuristic conversion. Algorithm SP is the algorithm for conversion. Second and third column are for the heuristic, while fourth and fifth are for the algorithm.*

the algorithmic version creates slightly better results. These come from later steps in the global placement when there is less overlap. During the early steps the heuristic produces better results. A more important matter is the run time comparison. The algorithmic legalization is twice as slow as the heuristic without delivering sufficient improvement to the placement.

Legalization algorithm setup Based on the previous discussion we choose to use the extended envelope based placement with diagonal heuristic method for legalization of placements during the remainder of the global placement experiments.

Fine-Tuning ω and δ

First we fine-tune ω and δ for the artificial nets. Experiments will show that these can have a great influence on the results of the global placement.

We let ω and δ vary among the values:

$$\begin{aligned} \omega &\in \{0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.7, 0.9, 1.1, 1.5\}, \\ \delta &\in \{0.1, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.7, 0.9\}. \end{aligned} \tag{9.1}$$

Since ω and δ are highly dependent we do placements for all 11×9 combinations of values.

Test circuits We have chosen a subset of the benchmark circuits that we believe represents the benchmarks well. The set include primary2, biomed, industry2, industry2g, avqlarge and clk. primary2 and biomed are medium-small sized standard-cell benchmarks. industry2g was chosen because of the general-cell property. industry2 was chosen to function as a reference to industry2g. avqlarge was chosen because it is a medium-large standard-cell benchmark. Finally clk was chosen as a representative of mixed-cell layout. Conducting the 99 test-runs for the larger of the circuits takes considerable time and this has limited the practical size of the circuits for this test.

Linearization or net-fraction scheme We would also like to test the linearization scheme against the net-fraction scheme (see section 7.1.1) during global placement. Therefore tests are conducted with both the linearization and net-fraction scheme.

Iterations As mentioned above the extreme number of tests makes it impractically to conduct large tests. This is also true when it comes to the number of iterations. We therefore limit these tests to a period of 20 iterations of the global placement.

Results The results of the these tests are displayed on figure 9.3 and 9.4. Here we have plotted a three-dimensional surface graph of netlength as a function of ω and δ for each circuit. The netlength shown is the *best* netlength after the 20 iterations.

The graphs shows four things:

- High values of the fall-off parameter δ do not function well. The δ-parameter determines how much a solution to the quadratic problem will be effected by previous solutions. A low value is likely good since it gives the heuristic more freedom.
- High values and low values of the initial artificial net-weight ω does not work well. This is not as easy to see from the graphs but inspection of the data revealed this.
- The net-size fraction and linearization schemes react more or less equally to the setup of ω and δ.
- All test-circuits seem to share good values of ω and δ.

To conclude further we have extracted the two best results of each of the circuits along with respective ω and δ parameters which are shown in table 9.3. Values are shown for both net-fraction and linearization. Except from the industry2 and industry2g circuits (which are related) the net-size fraction scheme delivers best results. The run times for a typical run ($(\omega, \delta) = (0.4, 0.4)$) is listed in table 9.4. The first iteration usually has higher run time since the initial solution to the quadratic problem must be determined. Run times are similar for all runs and for the large tests run times are severely smaller for the net-fraction scheme.

Choice of net-weight scheme Based on the fact that the net-fraction scheme has lower run time and produces the best results in most cases we choose the net-size fraction scheme for further testing.

	Selected ω-δ Results					
	Net-size fraction			Linearization		
Circuit	ω	δ	Netlength	ω	δ	Netlength
primary2	0.5	0.3	5,183,122	1.1	0.1	5,244,322
	0.35	0.35	5,186,529	1.1	0.2	5,278,800
	0.4	0.4	5,242,042	0.4	0.4	5,411,036
biomed	0.35	0.25	5,887,903	0.3	0.1	6,074,793
	0.5	0.5	5,909,397	0.3	0.3	6,036,830
	0.4	0.4	5,977,785	0.4	0.4	6,271,410
industry2	0.5	0.25	24,683,994	0.35	0.25	23,328,231
	0.3	0.3	24,714,195	0.45	0.2	23,601,899
	0.4	0.4	24,805,635	0.4	0.4	24,377,858
industry2g	0.5	0.1	29,240,737	0.2	0.1	28,094,528
	1.1	0.25	29,529,849	0.3	0.1	28,202,690
	0.4	0.4	30,850,407	0.4	0.4	30,858,771
avqlarge	0.1	0.25	11,890,982	0.3	0.1	12,067,708
	0.1	0.1	11,902,897	0.2	0.2	12,100,069
	0.4	0.4	12,869,293	0.4	0.4	13,943,764
clk	0.2	0.3	10,448,641	0.35	0.1	13,066,961
	0.1	0.2	10,472,650	0.2	0.2	13,240,889
	0.4	0.4	10,859,835	0.4	0.4	14,831,176

Table 9.3: *ω-δ results for the six circuits. The first two rows for each circuit are the two best results. The last low is simply for $(\omega, \delta) = (0.4, 0.4)$*

Choice of ω and δ Based on investigation of the graphs, the table of best results and the complete data-results we have decided to set:

$$(\omega, \delta) = (0.4, 0.4). \tag{9.2}$$

for further tests. The results of the $(\omega, \delta) = (0.4, 0.4)$ combination is also listed in table 9.3.

Number of Iterations

Based on these choices we now create the global placements without the clean-up step. We let the global placement heuristic run for 60 iterations on *all* benchmark circuits. This is done to investigate if the placement heuristic has any effect at all with many iterations. The results are shown in table 9.5. In general the results show that little improvement occurs after the 20th iteration and there is hardly any improvement after the 40th iteration.

Run-Times for ω-δ Experiment			
	Net-size fraction		
Circuit	1st. iteration	Average iteration	Total time
primary2	3.94	2.70	53.9
biomed	10.5	6.55	131
industry2	21.1	13.5	269
industry2g	21.3	13.8	275
avqlarge	86.9	41.4	828
clk	65.2	39.8	795
	Linearization		
Circuit	1st. iteration	Average iteration	Total time
primary2	6.72	3.98	79
biomed	17.5	10.4	208
industry2	45.9	22.7	456
industry2g	43.9	22.3	445
avqlarge	112	78.9	1579
clk	149	87.1	1742

Table 9.4: *Run times for the ω-δ experiment in seconds. The run times are from the $(\omega, \delta) = (0.4, 0.4)$-run but are similar for all runs. Run times are first iteration time, average iteration time and total time. Note that this is only for one of the 99 runs which were part of the ω-δ experiment.*

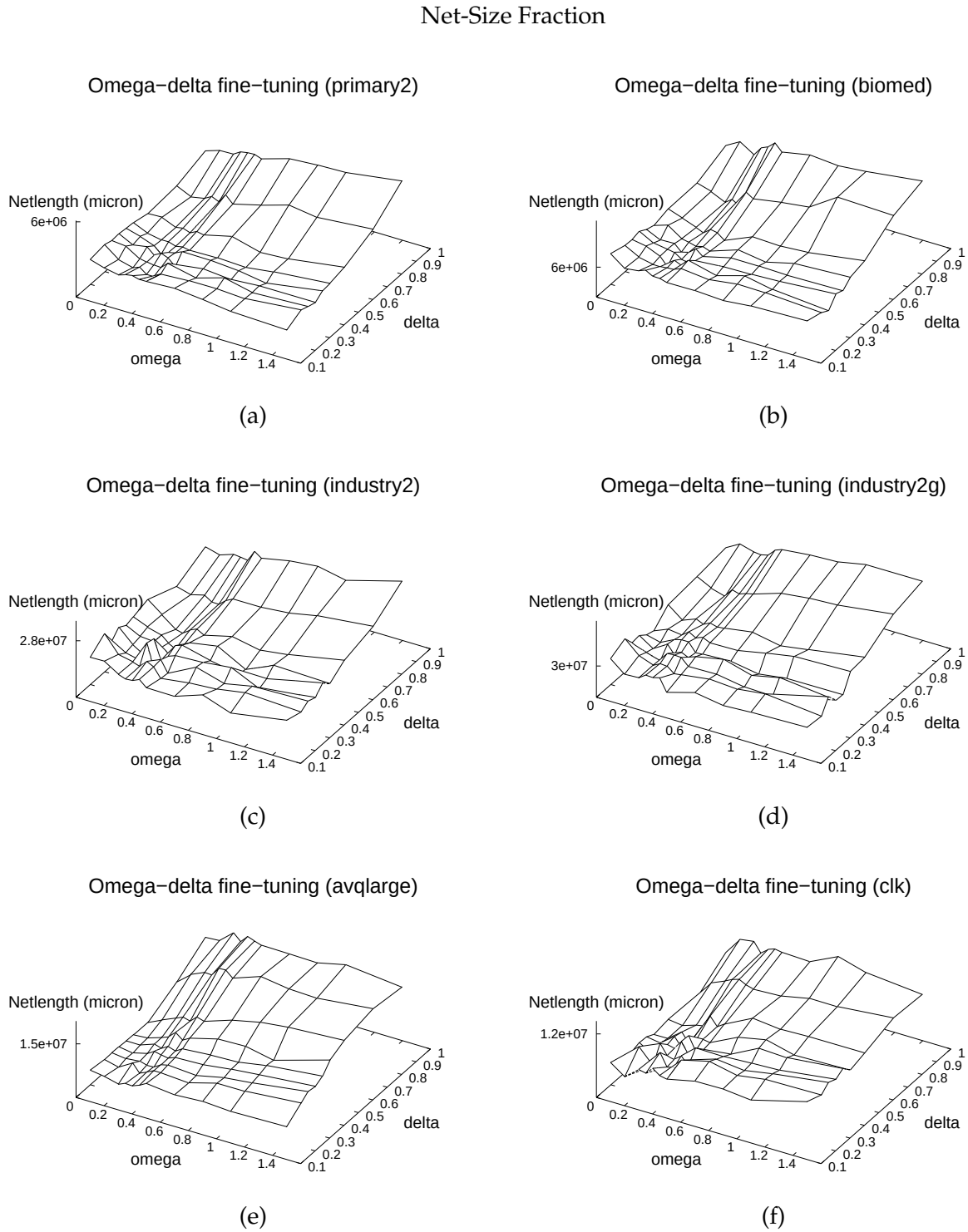


Figure 9.3: *Net-size fraction ω/δ-tuning. These graphs show how the netlength depends on ω and δ. The netlength is reported after 20-iterations using the specified ω and δ. The values of ω and δ were picked from the sets: $\omega \in \{0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.7, 0.9, 1.1, 1.5\}$ and $\delta \in \{0.1, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.7, 0.9\}$. For the large circuits the total time for these tests were as much as 20 hours.*

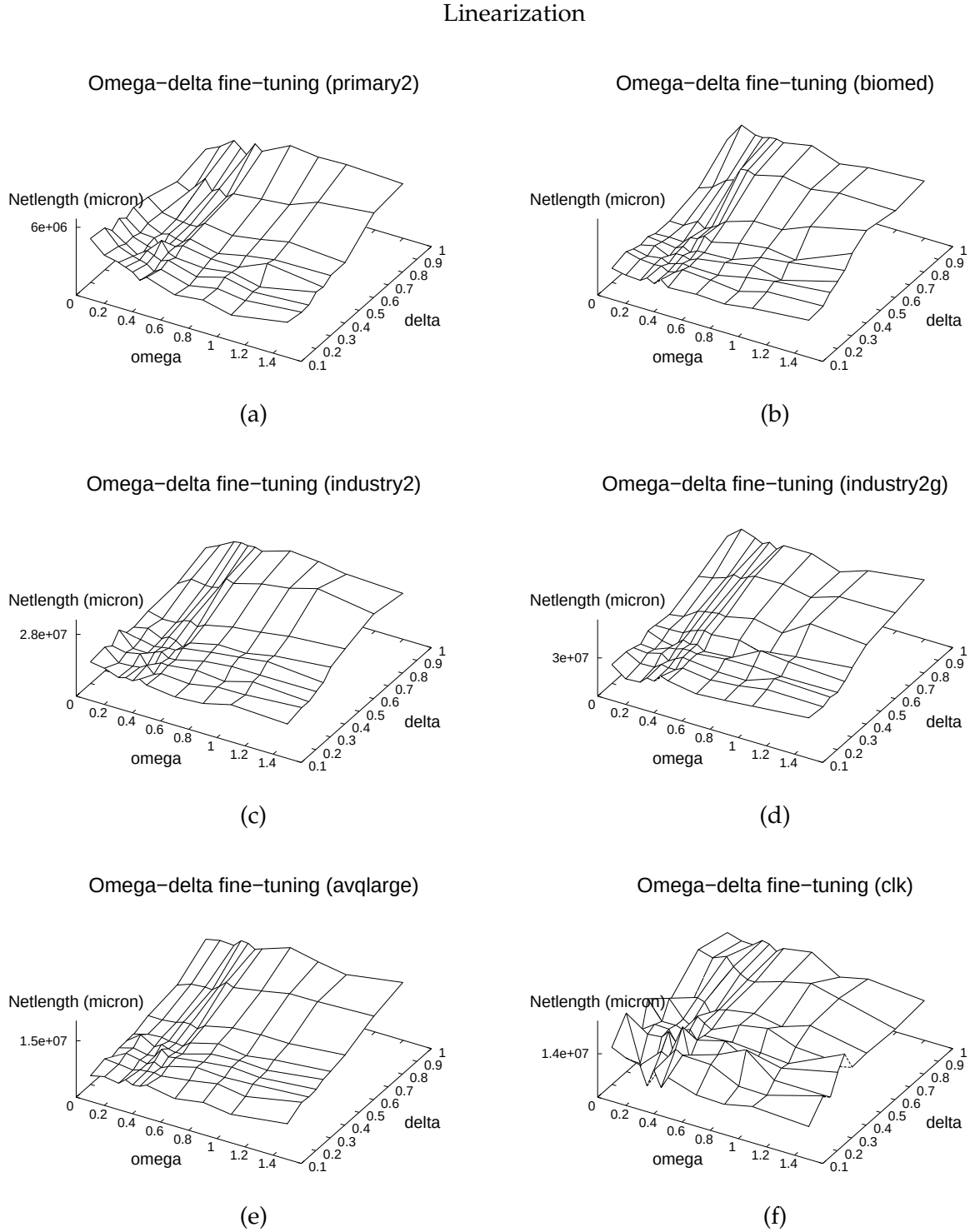


Figure 9.4: *Linearization ω/δ-tuning. These graphs shows how the netlength depends on ω and δ. The netlength is reported after 20-iterations using the specified ω and δ. The values of ω and δ were picked from the sets: $\omega \in \{0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.7, 0.9, 1.1, 1.5\}$ and $\delta \in \{0.1, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.7, 0.9\}$. For the large circuits the total time for these tests were as much as 20 hours.*

We also investigate how the heuristic behaves during the 60 iterations. We have plotted the development of netlength for 60 iterations of six circuits in the graphs off figure 9.5. The circuits are primary2, biomed, industry2, industry2g, avqlarge and decoder.

The graphs also show how the sequence-pair legalizer functions compared to the *standard-cell* legalizer. The sequence-pair legalizer is still used for creating artificial nets but the placement is *also* legalized using the standard-cell legalizer. The purpose of this is to investigate if the standard-cell legalizer is superior to the sequence-pair legalizer at later stages of the placement heuristic.

> **Choice of global placement iterations** From the graphs and table 9.5 we observe that global placement does not benefit from more than 40 iterations. Therefore we choose to use the results of the 40th iteration and we will also report total run times based on the 40th iteration.

The second element of the test; to compare sequence-pair legalization with the simple standard-cell legalization technique shows that the two techniques are more or less equally good. In general, however, the sequence-pair legalization technique seems slightly better during the first iterations. It should be no surprise that the two methods are almost equal in quality; both methods use the topology of the overlapping placement to legalize it.

9.3.2 Fine-Tuning Clean-Up

We now consider the clean-up step of global placement. For the clean-up step we must decide γ and the number of iterations.

We let clean-up use *simple* centered legalization. For the standard-cell instances we use the simple diagonal heuristic to determine sequence-pair. For the general-cell instances we use our placement-to-sequence-pair *algorithm*. This setup was determined through preliminary tests. Note that for the clean-up test of general-cell circuits we allow orientation change.

First we fine-tune γ. For this test we assume that γ should be equal on all circuits. The circuits *are* very similar, however γ has to do with overlap. A thorough investigation of γ would require that we conducted tests with all circuits. Instead we choose a limited number of circuits that we believe represent the benchmarks well. These are primary2, biomed, industry2, industry2g, industry3, avqlarge, decoder.

For this test we do 5 iterations of the clean-up step on the solutions from the 40th iteration of global placement on different values of γ. The value of γ is selected from the set

$$\gamma \in \{0.05, 0.1, 0.5, 1, 2.5, 5, 7, 10, 15, 20, 25, 30, 50, 70, 100, 150, 250\} \tag{9.3}$$

	Results of Global Placement without Clean-Up					
Circuit	Netlength 1 iteration	Netlength 10 iterations	Netlength 20 iterations	Netlength 30 iterations	Netlength 40 iterations	Netlength 60 iterations
industry1	2,313,980 1.71	1,975,538 13.93	1,888,261 26.1	1,888,261 38.3	1,871,481 50.7	1,788,286 75.8
industry2	39,001,191 13.5	29,297,628 91.4	24,805,635 155	24,492,458 215	24,314,034 276	24,314,034 402
industry3	115,969,220 26.3	85,151,301 198	73,602,786 340	73,574,522 478	73,574,522 616	73,574,522 896
avqlarge	23,490,822 65.8	15,167,572 452	12,587,219 732	12,587,219 1002	12,587,219 1275	12,587,219 1827
avqsmall	20,999,581 53.6	13,392,951 316	11,665,195 485	11,665,195 642	11,665,195 798	11,665,195 1114
biomed	8,375,860 6.65	6,497,541 46.5	5,977,785 76.7	5,977,785 106	5,977,785 135	5,977,785 196
golem3	339,017,223 168	295,221,580 1320	250,633,000 2155	248,758,003 2850	246,543,415 3545	246,199,030 4945
struct	2,044,071 1.82	1,329,956 14.4	1,278,283 26.8	1,243,151 39.3	1,237,178 51.9	1,237,178 77.5
primary1	1,558,346 0.60	1,362,124 5.0	1,328,566 9.49	1,323,310 14.9	1,307,988 18.7	1,307,988 28.0
primary2	6,904,771 3.20	5,524,673 26.1	5,242,042 48.5	5,242,042 70.7	5,176,233 93.2	5,176,233 138.8
primary2g	9,285,104 2.95	6,744,974 19.1	6,544,430 31.9	6,315,672 44.6	6,315,672 59.1	6,315,672 84.7
industry2g	51,438,650 14.1	36,016,646 95.8	30,850,407 160	30,267,954 224	30,267,954 288	30,267,954 420
industry3g	197,507,003 26.6	108,524,087 201	88,643,257 344	85,085,462 483	83,729,433 624	83,729,433 912
ami33k	159,933,561 68.1	152,619,873 466	145,380,403 972	145,380,403 1472	145,380,403 1977	145,380,403 3008
clk	13,068,885 55.9	13,063,269 396	10,292,641 729	10,292,641 1005	10,292,641 1341	10,292,641 1959
decoder	27,113,975 98.3	23,259,883 713	19,564,917 1337	19,564,917 1900	19,564,917 2471	19,564,917 3624
pu	191,728,197 379	176,729,527 2531	157,078,981 4745	157,078,981 6654	157,078,981 8572	157,078,981 12465

Table 9.5: *Global placement results before clean-up step at different iterations. First row of each circuit is netlength in micron. Second row is run time in seconds. Notice that the run time of the pu-circuit is extremely high compared to the remaining circuits.*

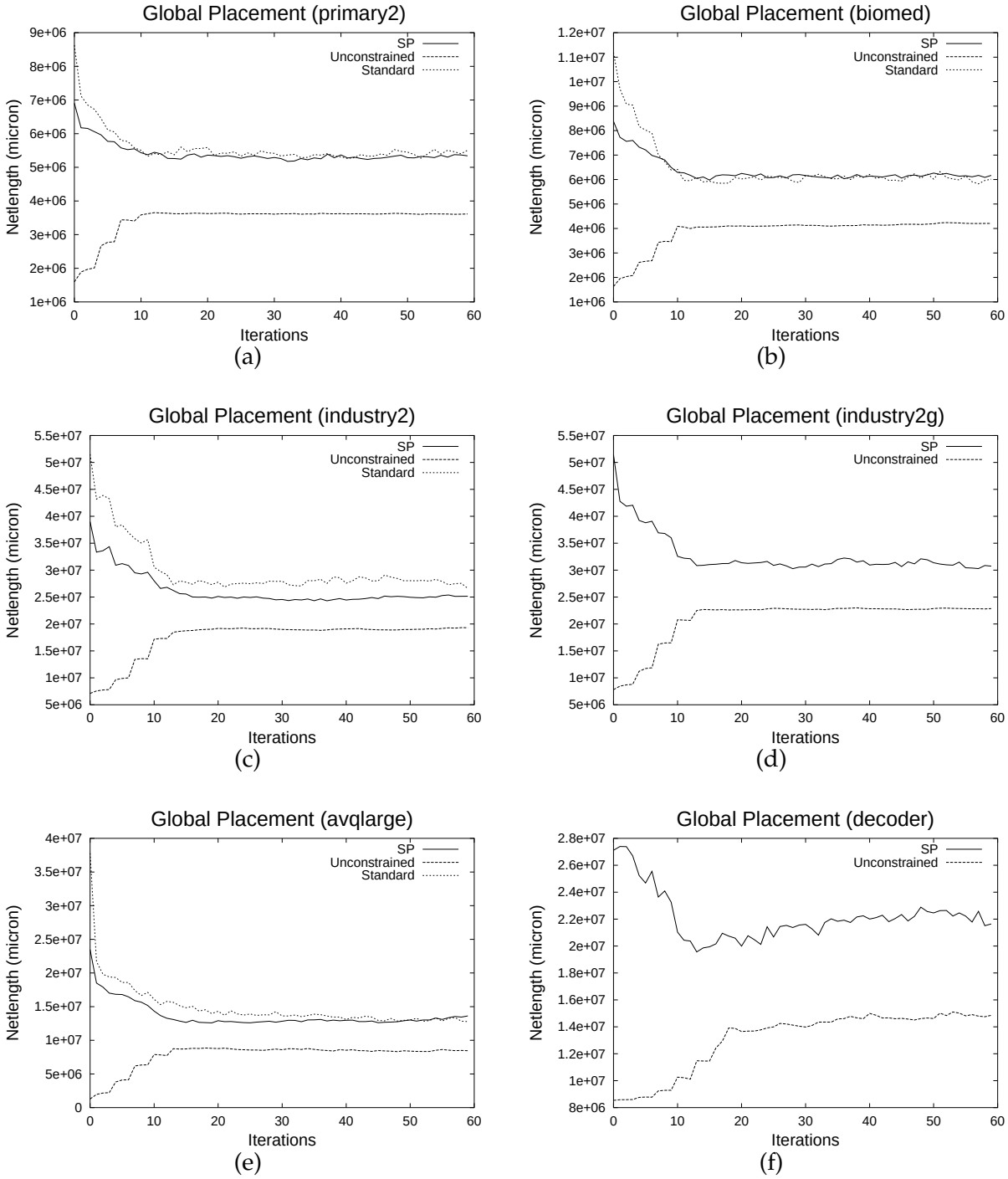


Figure 9.5: *Development of netlength during global placement. The netlength is shown during the first 60 iterations of global placement with $\omega, \delta = (0, 4, 0.4)$. Bounding-box netlength is reported for the unconstrained placement (unconstrained), the sequence-pair legalized placement (SP) and the standard-cell legalized placement (Standard). Note that industry2g and decoder are not standard-cell instances. Therefore there are no standard-cell legalization plots for these two circuits.*

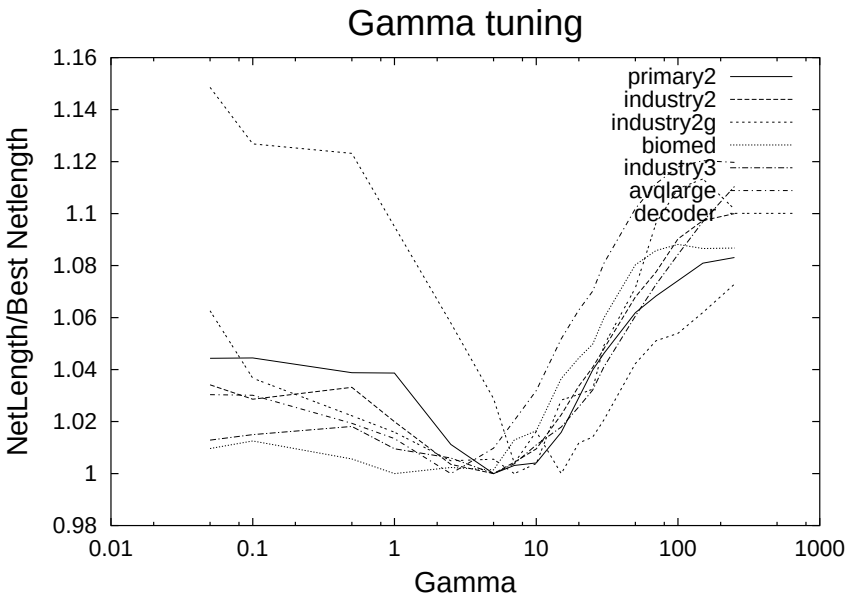


Figure 9.6: *Fine-tuning γ. 5 iterations of the clean-up step has been run for 17 different values of γ on seven circuits. The value plotted is the* best *netlength achieved during the 5 iterations divided with the best overall netlength achieved during the all 17 different runs.*

The test is conducted on the seven different circuits.

We have plotted the result of this test on the graph of figure 9.6. For each value of γ the netlength has been divided by the best overall netlength achieved during the test.

> **Choice of γ during clean-up** Based on the graph it appears that γ can be set to a global constant for all circuits and we deem that a good value of γ is 7 which will be used for further clean-up tests.

The second element of fine-tuning the clean-up step is to determine the number of sufficient clean-up iterations. For this we do 15 iterations of the clean-up step on three different circuits; primary2, industry2, decoder.

The table shows that there is little change in objective value after the 5th iteration.

> **Clean-up iterations** Based on results from table 9.6 we believe it is sufficient to do 5 clean-up iteration steps. Another reason to use few iterations steps is that the final placement heuristic may function better if the placement does not begin in a complete local-minimum.

9.3.3 Global Placement Results with Clean-Up

The results for the entire global placement step is enlisted in table 9.7.

	Clean-Up Tuning				
Circuit	1 iter.	3 iter.	5 iter.	10 iter.	15 iter.
primary2	4,470,178 1.26	4,231,582 3.55	4,168,973 5.8	4,143,416 11.34	4,143,081 16.9
industry2	20,738,792 9.0	19,608,424 30.75	19,445,688 44.1	19,319,476 78.0	19,294,304 105
decoder	15,082,699 65	13,458,868 148	13,061,748 183	12,988,976 385	12,906,141 556

Table 9.6: *Clean-up step development. For each circuit netlength and run time is reported. First row is netlength, second is run time in seconds.*

Results of Global Placement *with* Clean-Up							
Circuit	Netlength	Clean-up	Total	Circuit	Netlength	Clean-up	Total
primary1	1,101,678	1.12	19.9	primary2	4,168,973	6.0	93.2
biomed	4,566,847	31.0	166	struct	912,683	3.3	55.6
industry1	1,409,059	4.9	55.6	primary2g	4,737,722	14.8	102.5
industry2	19,529,866	38.4	314	industry2g	22,364,797	89.8	378
industry3	52,621,272	40.4	646	industry3g	59,525,034	116	740
avqsmall	8,294,763	894	1692	avqlarge	8,974,081	830	2105
golem3	167,805,962	327	5272	clk	8,509,108	99.5	1440
decoder	13,061,748	218	2689	pu	140,576,463	1050	9622
ami33k	132,442,156	397	2324				

Table 9.7: *Global placement results after clean-up. Columns are Netlength, clean-up run time in seconds, total time (global placement including clean-up) in seconds.*

Some things are worth noticing. First of all the run time is fine for the small to the medium sized circuits while the larger circuits, e.g. avqsmall, avqlarge, clk, decoder and pu have high-running times. They are still within hours however.

A second and *very* interesting thing to notice is the result of the general-cell circuits industry2g and industry3g. For these circuits the netlength is not much worse than for the plain standard-cell circuits industry2 and industry3. This is interesting because it demonstrates that the heuristic is well-suited for placing large instances of general-cells.

As a final note we discovered that the clean-up step does not function well on the ami33K circuit with $\gamma = 7$. Therefore we have determined a good value of γ to be 1000 by hand and this is used for this circuits. All other results were constructed based on our previous decisions.

	Forced Based Placement Results		
Circuit	Sequence-pair legalization	Standard-cell legalization	Running time 60 iterations
primary2	4,885,446	4,559,120	74
biomed	5,354,268	4,472,259	121
industry2g	25,068,031	-	245

Table 9.8: *Results of the force-based heuristic. The values of the second and third columns are netlengths of best legalization encountered. The run times to the right are for the force-based placement heuristic and not including legalizations which were done in a separate run. Run times are in seconds.*

9.3.4 Comparison with Force-Based Placement

The results of the previous section has little meaning without comparison with other global placement heuristics. Unfortunately we have found no reports for the global placement step of any other heuristics.

Instead we will use our own implementation of the force-based method of Eisenmann and Johannes [19] (see section 3.1.4) to compare our global placement method with their. It should be noted that several details are missing from [19]. Therefore we cannot guarantee that the implemented force-based heuristic has the same quality as the original. Secondly although we have tuned the heuristic to produce good results there is probably room for more fine-tuning.

For this experiment we use three circuits; primary2, biomed and industry2g. The three circuits are placed with the force-based global placement heuristic. We let the force-based heuristic run for 60 iterations. On the graphs of figure 9.7 we have plotted netlength for the 60 iterations. We report three different netlengths; unconstrained netlength (based on the current overlapping placement), netlength of a sequence-pair legalization of the unconstrained placement and netlength of a standard-cell legalization (see 7.1.2) of the unconstrained placement.

What can be seen from the graph is that our implementation of the force-based method use about 20 iterations to converge for the tested circuits. Also convergence seem better than for our global placement heuristic; the netlength is more stable. Finally the standard-cell legalization seem to outperform our sequence-pair legalization algorithm for both of the standard-cell circuits.

The best results of legalizations are reported in table 9.8. We also report the total running time for all 60 iterations. By comparing with table 9.5 it can be seen from table 9.8 that the force-based method is definitely superior to our global placement heuristic. The unconstrained placement of the 60th iteration of the force-based placement of primary2 is shown on figure 9.8. Please notice how little overlap there is in this placement.

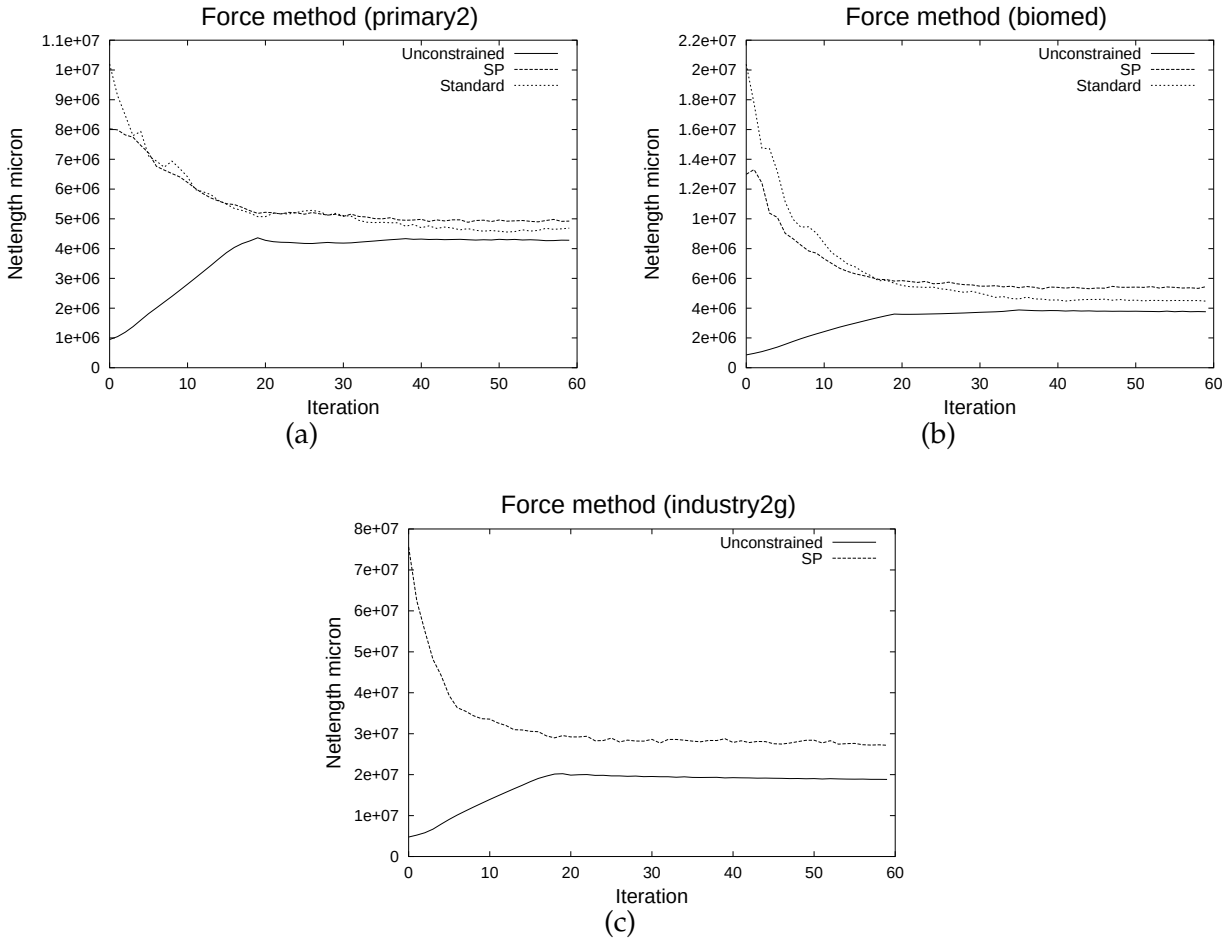


Figure 9.7: *Results of the force-based method. Please compare with figure 9.5. The reported netlengths are current unconstrained bounding-box netlength (unconstrained), current sequence-pair legalized bounding-box netlength (SP) and current standard-cell legalized bounding-box netlength (standard). Note that industry2g is not a standard-cell circuit so there are no results for the standard-cell legalizer for the circuit.*

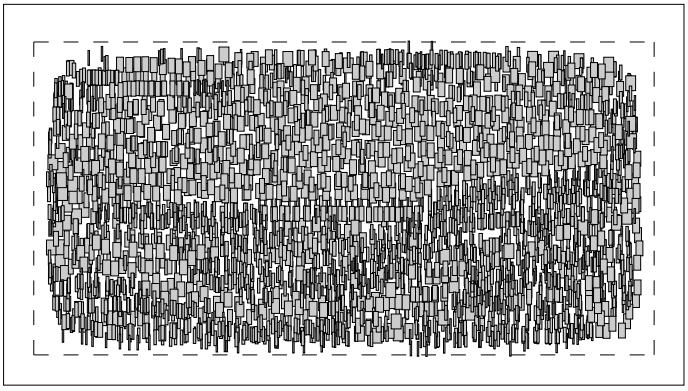


Figure 9.8: *The placement of 60th iteration of primary2 from the force-based method.*

	Clean-Up of Force-Based Placements	
Circuit	Clean-up Netlength	Clean-Up Time
primary2	4,342,362	5.0
biomed	4,411,510	25.2
industry2g	22,154,951	88.8

Table 9.9: *Clean-up of force-based results. The values of the second columns are netlengths. The third column is running time for clean-up step.*

Clean-Up of Force-Based Method

To test the quality of the solutions generated we also do a clean-up step for the force-based results. As initial solution of the clean-up step we pick the best sequence-pair legalization encountered during the 60 iterations of the force test. Otherwise the setup of the clean-up step is exactly as it was for our own heuristic.

The results of clean-up for the force-based solutions are given in table 9.9. The results are interesting. The clean-up step for the force-based results is not quite as effective as it were for our own results. Secondly the the solutions are very similar to our solutions after clean-up step. This also shows that the quality of a global placement cannot be determined by its netlength alone. It can only truly be determined by the subsequent optimization steps.

We will not discuss the force-based placement further but simply note that our global placement heuristic with the clean-up step could produce results which are comparable to the global placements of the force-based method for these three circuits. Of course it should be stressed that parameters of the implemented force-based method may not be optimally tuned and that the clean-up step is not tuned for the placements produced by the force-based heuristic. Although we expect the placements of our

	Standard-cell legalizer	
Circuit	Best result after 20 iterations	Time (20 iterations)
primary2	5,690,327	11.3
biomed	5,633,041	30.0
industry2	28,409,961	63.8
avqlarge	12,466,112	379

Table 9.10: *Second column gives best net-length results in micron of global placement based on standard-cell legalization for each of the circuits in the first column. Third column gives run times in seconds for the 20 iterations.*

method and the force-based method to be similar.

9.3.5 Comparison with Standard-Cell Legalization

As a final test of the global placement strategy we will also compare sequence-pair legalization with the standard-cell legalization technique. To do this we conduct tests on four standard-cell instances. We do 20 iterations of the global placement heuristic but with the standard-cell legalization technique as underlying legalizer instead of the sequence-pair legalization. The setup is otherwise as above (net-fraction and $(\omega, \delta) = (0.4, 0.4)$).

The results of the best placement after 20 iterations of the standard-cell legalization technique are shown in table 9.10

Two observation can be made from comparing table 9.10 with tables 9.5 and 9.4.

- The standard-cell legalizer will in some cases produce better results than the sequence-pair legalizer.
- The standard-cell legalization based global placement is several times faster than the sequence-pair based.

Both of these observations should not come as a surprise. The standard-cell legalization is similar to the sequence-pair legalization in that both use the topology of the overlapping placement to construct a legal placement. Also the sequence-pair legalization suffers from the varied α strategy which in essence means that it does 20 legalizations in each iteration. We should stress however that the *standard-cell legalization technique is incapable of handling mixed- and general-cell layouts*. We will not pursue the subject of standard-cell vs. sequence-pair based legalization further in this thesis.

9.3.6 Conclusion on Global Placement

Our global placement heuristic produces fine results combined with the clean-up step. The heuristic *is* slow for large instances but small instances are handled well.

In general the clean-up step has proven very effective and in all cases faster than the basic 40 iterations of global placement.

Our heuristic seems comparable to the implemented force-based method with clean-up step. However without clean-up step the force-based method is superior.

The heuristic with clean-up step is also quite capable of placing general-cell circuits of substantial size (industry2g and industry3g). The general-cell circuits were placed with netlengths comparable to those of the corresponding standard-cell circuits (industry2 and industry3).

As a conclusion the global placement heuristic could certainly function as a prototype for a full-blown global placement heuristic. However we deem that the current implementation is still too rough around the edges for practical use.

9.4 Experiments for Final Placement

In this section we conduct experiments with final placement. The final placement heuristic has many variables which can all influence it severely. Therefore it is crucial that we conduct tests that determine good values for the parameters.

Our final placement is controlled by simulated annealing. This alone introduces several variables and also a greater amount of randomness to the solutions. In general for heuristics which rely on randomness it is necessary to conduct *many* experiments to determine the behavior of the heuristic.

Unfortunately time and computational power does not allow us to conduct a complete range of tests on all circuits. Therefore we will rely on stability of the heuristic. We will determine if the heuristic produces close to equal results with equal parameter values but different random seed which should give some indication of stability.

The section is otherwise similar to the section of global placement. First we will fine-tune a subset of the many parameters of final placement. Then we will give the results of *our* final placement. Finally we will compare the results with results of other authors.

legalization setup For legalization we will use the placement-to-sequence-pair algorithm since it generates better placements when there is little overlap. The legalization setup is otherwise as it was for the clean-up step.

9.4.1 Fine-Tuning Final Placement

To recapitulate the parameters of final placement are:

- **Sub-circuit size** k The size of the sub-circuits extracted by relaxation based local search.
- **Overlap penalty** γ The overlap penalty determines how much overlap is allowed in local search solutions.
- **Fraction of modules moved** q The amount of modules to move before legalization.
- **SA parameter – initial temperature** t_0 The initial temperature of the simulated annealer.
- **SA parameter – cooling factor** The temperature t drops by $t = K \cdot t$ whenever a move is accepted.
- **Reset interval** r The time between resets of the temperature of the simulated annealer.

Through preliminary tests we had determined that a good and practical value of the sub-circuit size is 10 (see 8.4.2). We will consider adjustment of the remaining parameters however.

Tuning Simulated Annealing

We begin by tuning the simulated annealing constants since good values of the remaining parameters may depend on the behavior of the simulated annealer.

Our simulated annealing has only two parameters; initial temperature and cooling factor. Just as the other parameters of our placement heuristics good values may be highly dependent on the placement instance. However based on the discussion of section 5 we assume that good values are equal for all the circuits for this test.

Unfortunately t_0 and K are highly dependent. Therefore we conduct tests with 28 different combinations of t_0 and K. The values are taken from the sets:

$$t_0 \in \{10, 50, 100, 500, 1000, 5000, 10000\} \quad and \quad K \in \{0.99, 0.999, 0.9999, 0.99999\}$$

To account for randomness we do 5 different runs with each combination. Each run takes *10 minutes* (total time: 1400 minutes! for each circuit). Because we need to determine the behavior of the simulated annealer during a longer period, not just for the first iterations, we will conduct the tests on small circuits. The hope is that the small circuits will share sufficient properties with the larger circuits that the behavior of the heuristic is almost equal.

Also since the choice of t_0 and K influence the behavior of the final placement heuristic we expect good values to work well for both general-cell and standard-cell instances.

	SA-parameter Test Results				
Circuit	T_0	K	Min	Average	Max
struct	500	0.99999	793,485	795,953	798,812
	1000	0.9999	796,441	798,068	802,503
	5000	0.9999	785,382	791,412	795,681
industry1	500	0.9999	1,240,314	1,261,288	1,277,173
	100	0.99999	1,258,289	1,263,796	1,267,071
	1000	0.9999	1,287,825	1,292,801	1,303,946
primary2	1000	0.9999	3,888,819	3,896,796	3,917,124
	500	0.9999	3,898,440	3,917,492	3,927,806

Table 9.11: *Results of tests of simulated annealing parameters. The second and third columns are values of t_0 and K. The last three columns are netlengths of respectively best, average and worst results for each combination of t_0 and K. The combinations shown are the two best average results and our choice of t_0 and K. Note there are only two results of primary2 since the combination $(t_0, K) = (1000, 0.9999)$ was also the best among the average results.*

The circuits we have chosen are struct, industry1 and primary2. The smallest of these circuits contains 1888 modules and the largest 2907. For the γ parameter we have chosen a value of 10 which is a pessimistic guess based on the γ fine-tuning of global placement (see section 9.3.1). For the fraction of moves to be accepted in each legalization iteration we have chosen 15 % based on some initial testing.

The results of the tests are plotted on figures 9.9 and 9.10. We have plotted the minimum, average and maximum best netlength achieved during the 5 runs for each combination of the two variables.

First of all we note how similar the graphs of min, max and average are for each individual circuit. This points in the direction that the simulated annealing is stable and that most work is probably done by the relaxation based local search.

Secondly the graphs have equal structure which we assume comes from by the fact that t_0 and K have more or less same influence independently of the circuits. The results of the combinations with best *average* results for each circuit are also presented in table 9.11. Please note how little variation there is between min, max and average values of each combination of t_0 and K .

Choice of t_0 and K Based on these graphs and further data inspection we have determined that good values for t_0 and K are $t_0 = 1000$ and $K = 0.9999$.

Tuning γ and q

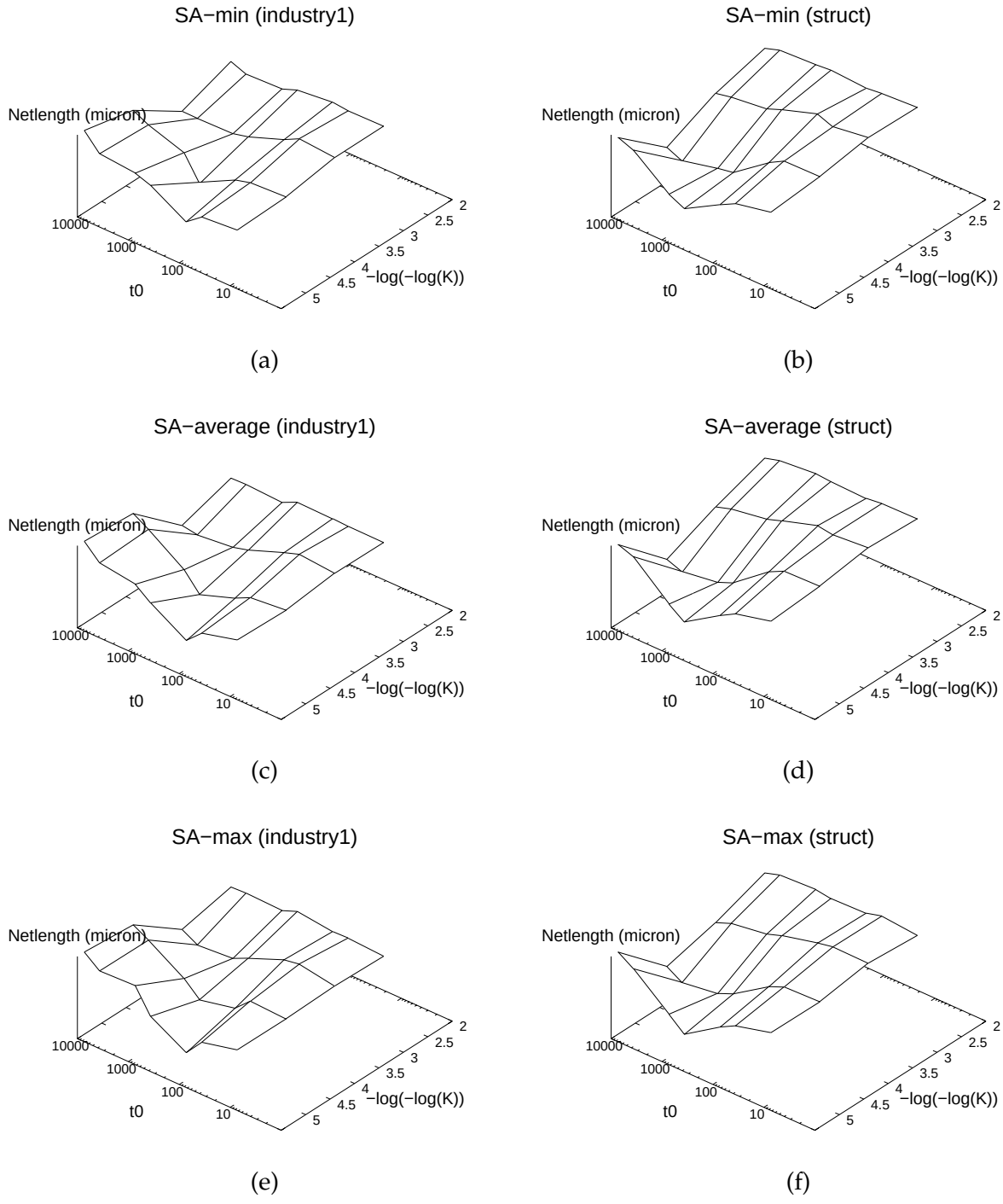
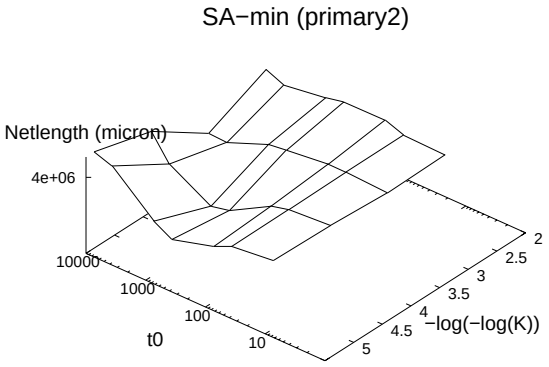
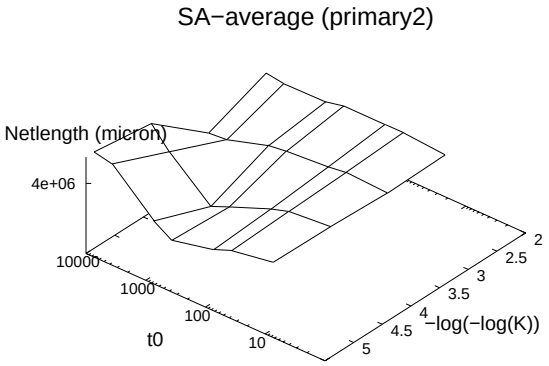


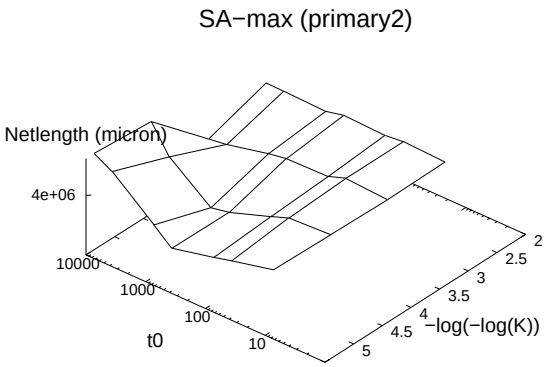
Figure 9.9: *Plots of t_0 and K of the simulated annealer for industry1 and struct. Please note that K is among the values $\{0.99, 0.999, 0.9999, 0.99999\}$ so e.g. $-\log(-\log(0.9999)) = 4.3622$. The top row plots are of the best results achieved during all 5 runs. The second row plots are of the average results achieved and the final row plots are of the worst (max) results achieved.*



(a)



(b)



(c)

Figure 9.10: *Plots of t_0 and K of the simulated annealer for primary2. Please note that K is among the values $\{0.99, 0.999, 0.9999, 0.99999\}$ so e.g. $-\log(-\log(0.9999)) = 4.3622$. The top row plots are of the best results achieved during all 5 runs. The second row plots are of the average results achieved and the final row plots are of the worst (max) results achieved.*

The next element of tuning the final placement heuristic is determining γ and the fraction of moves required between each legalization iteration q. These two variables are likely highly correlated since the number of moves limits the amount of total overlap allowed while γ limits the amount of overlap allowed for each step of the relaxed local search.

For this test we use 6 circuits. The circuits are primary2, industry2g, industry2, decoder and clk. Once again we search for a combination of good values. γ and q are selected from the sets:

$$\gamma \in \{7, 10, 15, 35, 50\}, \qquad q \in \{0.05, 0.10, 0.15, 0.25\}. \tag{9.4}$$

We have discovered that for the two IBM-circuits the limited values of γ did not produce good results. Therefore we *also* test for $\gamma \in \{100, 1000\}$ for these circuits.

We do not wish to set $q > 0.25$ since we deem that no more than 25% of the modules will move during later iterations.

The parameters of the test are based on the results of the simulated annealing as determined in the previous section. We let the test of the smaller circuits primary2, industry2 and industry2g run for *10 minutes*. This should be sufficient time for the heuristic to do several legalization iterations. For the larger circuits, clk and decoder, we deem that *20 minutes* is necessary. For each of the small circuits the total time of this test is 200 minutes. For the larger circuits it is 560 minutes.

We include industry2g in the list since this is a general-cell circuit which may be more sensitive to the correct choice of γ since it is harder to place.

At this point we *assume* that the final placement is stable enough that the results will represent general tendencies. This assumption is based on the tests of the simulated annealing parameters which showed high stability.

The results are plotted on the graphs of figure 9.11. The best and second best combination of each circuit are listed in table 9.12.

Firstly the graphs show that the value of γ for the IBM-circuits must be very high. Also by careful inspection of the graphs it may be seen that the heuristic generally performs best with $q = 0.15$ and $q = 0.25$. Secondly it can be seen that the IBM-circuits require a very high γ value. In fact values of γ less than 1000 results in unstable behavior where the heuristic does not reduce the netlength of the clean-up step. Thirdly it can be seen that the correlation of γ and q is less than could have been expected. A good value of γ generally performs well independently of q.

Choice of γ and q. Based on the graphs and table 9.12 we choose $(\gamma, q) = (15, 0.25)$ for the MCNC circuits and $(\gamma, q) = (1000, 0.25)$ for the IBM-circuits since these produce the best results according to table 9.12.

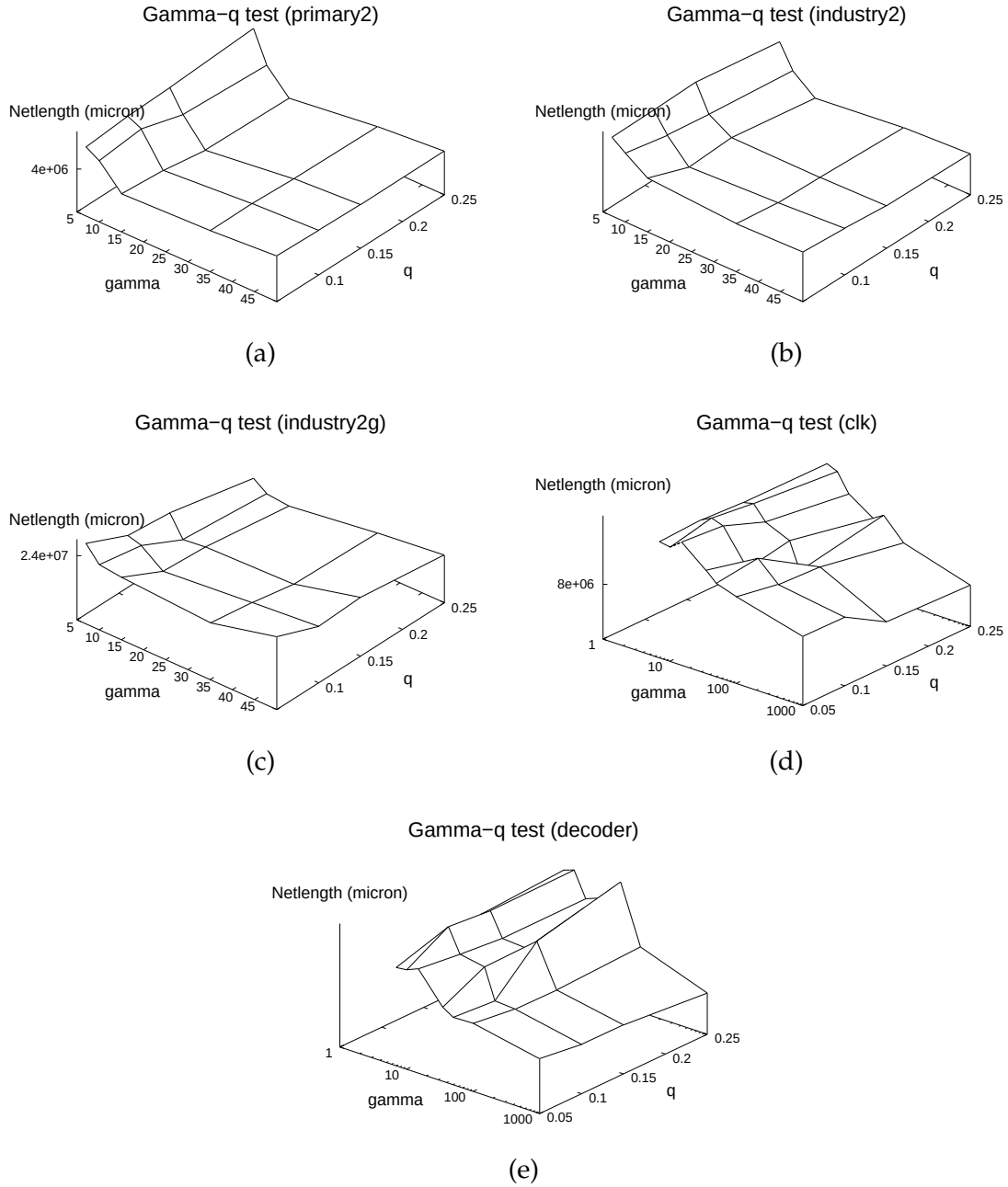


Figure 9.11: *The graphs of γ and q variation for the five test circuits. Notice that the two IBM-circuits require a very high γ-value.*

	γ-q Test Results		
Circuit	γ	q	Netlength
primary2	15	0.25	3,880,407
	15	0.15	3,890,716
industry2	15	0.25	18771364
	35	0.25	18971844
industry2g	15	0.25	22,711,505
	10	0.25	22,796,868
clk	1000	0.25	7,873,715
	1000	0.15	7,901,372
decoder	1000	0.25	12,762,874
	1000	0.15	12,876,283

Table 9.12: *Results of the test with combination of γ and q. The reported results are best (first row) and second best (second row) for each combination.*

	Reset Test Results					
Circuit	No reset	1 iter.	5 iter.	10 iter.	25 iter.	50 iter.
primary2	3,891,247	3,936,387	3,920,714	3,935,509	3,895,359	3,838,567
industry2	18,292,890	18,177,942	18,139,355	18,171,417	18,151,144	18,292,890

Table 9.13: *Results of the reset tests. Columns are with reset after specified number of iterations. Best netlengths after 1 hour is reported for each circuit with the 6 different reset intervals.*

Comment on q It may be that the reason the heuristic performs best with $q = 0.25$ is that it does fewer legalizations and therefore has more time to optimize. This could explain why the heuristic performs almost equally with $q = 0.15$.

Tuning Reset Interval

The final parameter we will test of the final placement is the number of legalization iterations between reset of the heuristic. This is a hard value to determine. If set too high the heuristic will fall into a local minimum and never escape. If set too low the final placement will never descent into local-minimum.

We conduct this test on only two circuits; primary2 and industry2. Again we assume that the heuristic will behave relatively equal on all circuits and since this has to do with the simulated annealing we expect equal behavior for the standard- and general-cell circuits also. We set the parameters as above and let the heuristic run 1 hour on the two circuits for different values of r. We choose r from the set $r \in \{\infty, 1, 5, 10, 25, 50\}$ (∞ means no reset). The results of the two tests are shown in table 9.13.

	Final Placement of Small Standard-Cell Circuits					
Circuit	Clean-up	1 min.	5 min.	10 min.	30 min.	60 min
primary1	1,101,678	1,055,982	1,024,931	1,018,437	1,011,761	994,349
struct	912,683	849,726	804,643	795,115	787,087	785,655
industry1	1,409,059	1,331,902	1,293,420	1,293,420	1,293,420	1,293,420

Table 9.14: *Netlength results of the small MCNC standard-cell circuits during final placement. Clean-up column is the netlength after clean-up step.*

The results are not conclusive. For the primary2 circuit it seems that a reset interval of anything less than 50 is too short. For the industry2 circuit we should note that the 50th iteration is never reached for test of the 50 iterations reset interval. Therefore less than 50 iterations may be necessary for the industry2 circuit, so that it does not fall into a local minimum without ever resetting. The best result for the test involving industry2 is achieved with an interval of 5-10 iterations.

> **Reset interval choice** To accommodate for the inconclusive results of the reset test we reset every 50th legalization iteration for the small circuits (primary1, primary2, struct, industry1) and every 10th iteration for the larger circuits (all other).

Comment on choice of reset interval The test is non-conclusive and to determine the perfect reset interval more tests would need to be conducted. Unfortunately time did not allow us to do this.

9.4.2 Final Placement Results

We now present the final placement results with our final placement heuristic parameter values defined according to the previous discussion. We have divided the benchmark circuits into several groups.

Small instances The first group consists of the small standard-cell circuits primary1, struct and industry1. For these circuits we let the final placement heuristic run for 60 minutes. The results are listed in table 9.14.

The results are certainly promising. Most improvement seem to lie within the first half hour. Unfortunately the placement heuristic does not improve the industry1 circuit after 10 minutes. This could be explained with the fact that the placement heuristic is not completely tuned for the industry1 circuit.

	Final Placement of Medium Sized Standard-Cell Circuits					
Circuit	Clean-up	5 min.	10 min.	30 min.	60 min.	3 Hours
primary2	4,168,973	3,969,689	3,922,070	3,866,628	3,838,567	3,795,604
biomed	4,566,847	4,274,733	4,159,376	4,009,024	3,945,261	3,843,981
industry2	19,529,866	19,055,008	18,873,235	18,457,240	18,171,417	18,073,560
industry3	52,621,272	51,694,906	50,517,583	48,997,745	48,669,067	48,327,737
avqsmall	8,294,763	8,294,763	8,294,763	8,294,763	8,134,807	7,853,757
avqlarge	8,974,081	8,974,081	8,974,081	8,974,081	8,768,265	8,500,176

Table 9.15: *Netlength results of the medium sized MCNC standard-cell circuits during final placement. Clean-up column is the netlength after clean-up step.*

Medium sized instances The second group consists of the medium sized standard-cell circuits primary2, biomed, industry2, industry3, avqsmall and avqlarge. For these circuits we let the placement heuristic run for 3 hours. The results are listed in table 9.15.

For the two smaller circuits primary2 and biomed the heuristic seems to perform well and there is improvement even during the last 2 hours although most improvement lies within the first our. For the slightly larger circuits, industry2 and industry3, improvement is good. Once again most improvement is within the first hour. The results are not as promising for the final two circuits. Here the placement heuristic does not improve on the initial solution from the clean-up step for the first half-hour. This could be explained by the fact that the placement heuristic is not tuned for the two avq circuits and that the simulated annealer is accepting many non-improving moves at random.

General-cell instances The third group consists of the general-cell circuits industry2g, industry3g and ami33k. Although they share order of modules with the medium sized circuits, different orientations are allowed during the local search, and we deem that they require double running time to compensate for that. Therefore we let the placement heuristic run for 6 hours on these circuits. The results are listed in table 9.16

The results for the three standard-cell based instances are quite good. The primary2g has 20% higher netlength than the corresponding primary2 circuit. The industry2g has only 13 % higher netlength than its standard-cell counterpart in spite of the fact that it is a larger instance. The industry3g has netlength only 11% higher netlength than industry3. There is little improvement after the first hour and it appears that the general-cell circuits share running time with the standard-cell circuits. From this we can conclude that the extra solution space due to orientation search does not slow the heuristic down considerably. Also improvement of the industry2g and industry3g circuit are same order of magnitude as it were for industry2 and industry3.

	Final Placement of General-Cell Circuits					
Circuit	Clean-up	10 min.	30 min.	60 min.	3 Hours	6 Hours
primary2g	4,737,722	4,561,091	4,559,165	4,556,133	4,544,611	4,544,611
industry2g	22,364,797	21463804	20726576	20414968	20338670	20338670
industry3g	59,525,034	57,879,072	56,079,329	55,540,273	54,959,702	54,959,702
ami33k	132,442,156	132,442,156	133,138,107	132,588,774	132,010,166	132,010,166

Table 9.16: *Netlength results of the general-cell circuits during final placement. Clean-up column is the netlength after clean-up step.*

	Final Placement of Large Circuits					
Circuit	Clean-up	10 min.	60 min.	3 Hours	6 hours	9 Hours
golem3	167,805,962	165,376,161	163,742,469	161,338,611	160,219,553	-
clk	8,509,108	8,234,521	7,892,079	7,855,217	7,855,217	7,855,217
decoder	13,061,748	13,026,466	12,333,089	11,950,597	11,810,612	11,740,484
pu	140,576,463	140,576,463	140,576,463	136,529,938	128,231,982	128,508,781

Table 9.17: *Netlength results of the large standard- and mixed-cell circuits during final placement. Clean-up column is the netlength after clean-up step. Note that the golem3 circuit is only run for 6 hours.*

Unfortunately the ami33k circuit has not been improved substantially during final placement. It could be that our final placement heuristic is geared mostly towards instances with fewer nets on each module or that the global placement of ami33k is already quite good (see figure 5.6) and that our final placement can not improve it. From [24] we know that there exist solutions to the ami33 circuit with netlengths in the area of 40- 50.000 micron. Multiplying these values by 1024 we get about 50.000.000 micron so it appears that we are quite a distance from the optimal placement.

Large instances The fourth group consists of the large standard-cell circuit golem3 and the large mixed-cell IBM-circuits. For the golem3 circuit we the let the final placement heuristic run for 6 hours. For each of the IBM-circuits we let the final placement heuristic run for 9 hours. The results are listed in 9.17.

The results are not as promising as for the smaller instances. Improvement still occurs in the last three hours for all circuits except clk. clk does not improve for the last eight hours which clearly shows that the final placement heuristic is not tuned for the clk instance. A similar observation holds for the pu circuit which does not improve for the first 3 hours. Inspection of the results during this time revealed substantial "jumps" in netlength which could mean that the simulated annealing is still in its early stages or that the γ-value is not set high enough.

9.4.3 Development of Final Placement

Before comparing our results with those of other authors we briefly consider the development of solutions during the final placement heuristic over time.

The purpose of this is to illustrate how the final placement works on the placements. Therefore we have chosen a subset of circuits for which the previous results were good and one for which the results were poor. The circuits we have chosen for this demonstration are primary2, industry2, industry2g and clk.

On the graphs of figure 9.12 we have plotted the development of the netlength over time for these four circuits. The netlength is reported at each legalization iteration. The unconstrained netlength before legalization is also reported.

We suspect that the large frequently occurring jumps in netlength are due to the reset strategy. Under this assumption the graphs clearly shows that the reset interval strategy does not work well for the placement circuits. Our assumption for the reset strategy was that it would not increase netlength too much. This assumption holds. At no time does the netlength seem to increase beyond its value of the first iterations.

The heuristic behaves best on the industry2 circuit because descent is allowed over time. However it does not seem as though the circuit ever enters a local minimum state. The other three circuits are constantly interrupted during their descents which must mean that the reset interval is too short and results in too large changes in netlength. This explains the poor result of the clk circuit; the heuristic actually comes close to improvements but the complete descent is always interrupted.

The most surprising result is that the heuristic for industry2 and industry2g does not seem to share behavior although they have almost common properties. This could indicate that it is very hard to determine a good reset interval.

9.4.4 Comparison with Other Final Placement Results for the Circuits

Before concluding on *our* results we first present results of other writers. The specific instances of the MCNC-benchmarks originates – as described in 5.1.4 – from [24]. Here the circuits had been optimized with the TimberWolf Commercial[12] v. 1.2 placement tool and the placement tool XQ developed at IDMUB. The results of these previous methods are enlisted in table 9.18 and are from [24].

No run times are given for the TimberWolf (TW) placement tool. The XQ placements were conducted on an Intel Pentium II 500 Mhz. The stand-alone guided local search (GLS) approach was based on a very simple global placement tool. GLS [24] was also tested as improvement heuristic on the TimberWolf and XQ placements. The "best

[12]The commercial version numbering differ from the numbering in the articles and the commercial versions also succeeds those of the articles.

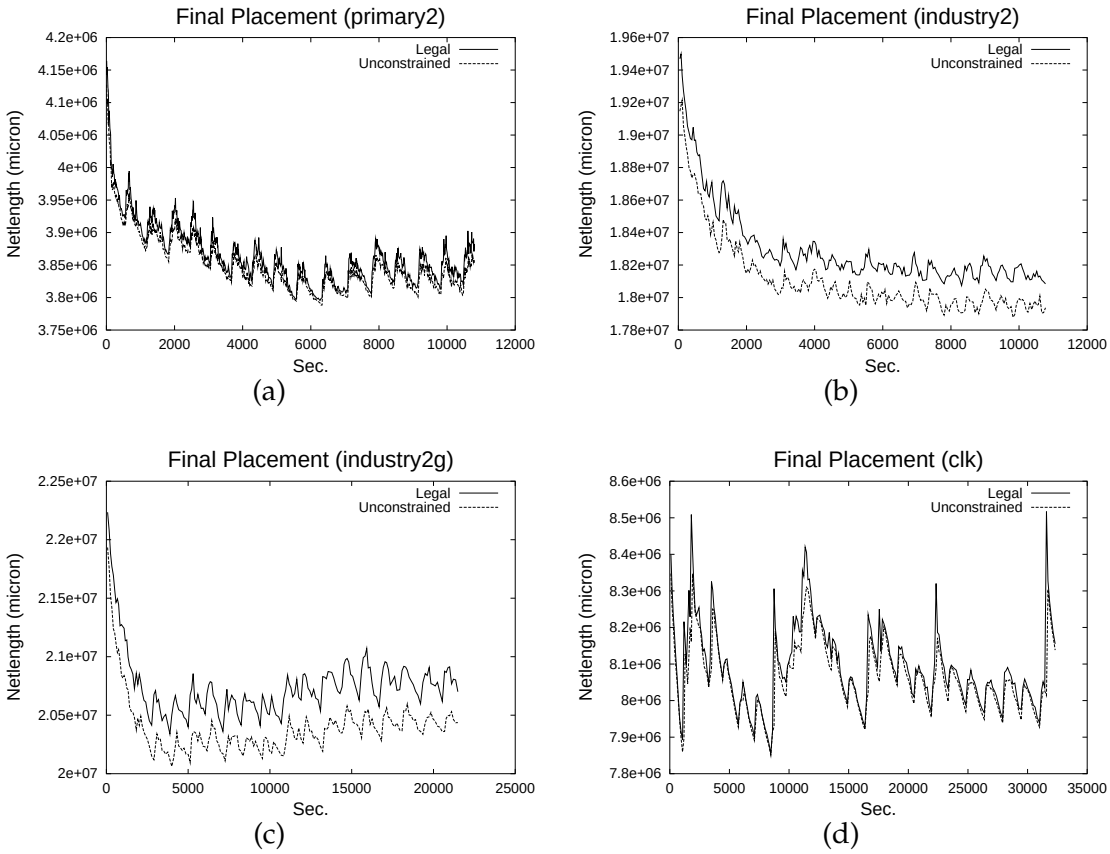


Figure 9.12: *Development of the final placement solutions over time for the four circuits primary2, industry2, industry2g and clk. The 'legal' value is the legal solution at this time as return by the legalization step. The 'unconstrained' value is the netlength of the placement before legalization.*

	Results of Other Final Placement Heuristics				
Circuit	TW	XQ	Run time for XQ	Stand-alone GLS	Best GLS
struct	778,321	774,289	54	837,699	678,152
primary1	987,141	1,137,756	31	999,483	949,353
industry1	1,866,177	1,747,234	31	1,699,490	1,634,801
primary2	3,637,653	4,096,888	150	4,460,830	3,612,800
biomed	3,467,190	4,320,451	1254	6,044,120	3,442,250
industry2	14,455,858	16,906,936	1,142	22,925,755	14,288,855
industry3	42,652,420	48,740,536	1,260	75,425,997	42,582,937
avqlarge	6,877,290	8,546,128	1320	32,643,082	6,786,482
golem3	118,572,063	118,686,359	8400	315,954,658	113,514,220
clk	-	5,286,747	-	-	4,903,923
decoder	-	7,781,409	-	-	7,022,407
pu	-	62,268,385	-	-	52,414,317

Table 9.18: *Results of other final placements of the specific MCNC instances that we use (See text)*

GLS" results are the best of the improvement results. The stand-alone GLS and "best GLS" run-times are 3 hours for the MCNC-circuits and 12 hours for the IBM-circuits on an Intel Pentium III 800 MHz.

We should note that one cannot compare TimberWolf and XQ with this table. Firstly XQ is geared towards large standard-cell instances with macros. Secondly no run-times are reported for the TimberWolf heuristic. Thirdly XQ may also optimize other objective functions than netlength (e.g congestion).

Also please note that several of the placements after our clean-up step from table 9.7 are actually quite close or better than some of the reported placements with run times that are comparable to those of XQ, but our tests are conducted on a machine with double clock frequency.

We now compare these results with ours. We have listed all circuits in table 9.19 along with the difference in percent between our final placement results, our initial results, the results of TimberWolf and the results of XQ. We have not listed the results of the GLS heuristic because the interesting results of GLS are improvements of the TimberWolf and XQ placements.

Discussion on improvement For all circuits except ami33k the final placement improves the solution of the clean-up step with between 4 and 10 %. From table 9.6 we know that at least primary2 and industry2 are close to a swap-neighborhood minimum after clean-up. The relaxation based local search of our final placement can easily produce solution below the values of table 9.6. We feel that the $5-10\%$ improvements

Comparison with other authors							
Circuit	Clean-up Solution	TimberWolf	XQ	Circuit	Clean-Up Solution	TimberWolf	XQ
primary1	90.3 %	100.7 %	87 %	primary2	91.0 %	104.3 %	92.6 %
biomed	84.1 %	110.8 %	88.9 %	struct	86.1 %	100.9 %	101.4 %
industry1	91.8 %	69.3 %	74.0 %	primary2g	95.9 %	-	-
industry2	93.0 %	125.0 %	106.9 %	industry2g	90.9 %	-	-
industry3	91.8 %	113.3 %	99.2 %	industry3g	92.3 %	-	-
avqsmall	94.7 %	-	-	avqlarge	94.7 %	123.5 %	99.5 %
golem3	94.5 %	135.1 %	135.0 %	clk	92.3 %	-	148.6 %
decoder	90.0 %	-	167.2 %	pu	91.4 %	-	206.4 %
ami33k	99.7	-	-				

Table 9.19: *Comparison with other authors based on table 9.18. The Clean-Up columns is comparison with the clean-up initial solution. Percentages are calculated as* $\frac{\text{final placement result}}{\text{other result}}$ *so less is better.*

during final placement are quite good considering that this is a preliminary new placement method and that the simulated annealing method has not been well-tuned.

Comparison with TimberWolf and XQ For the small and medium sized circuits our final placement results are certainly comparable with both XQ and TimberWolf. The worst of these lie within 25 % percent of the results of XQ and TimberWolf.

For the industry1 circuit the result is significantly better than either XQ and TimberWolf. Based on the fact that the result of industry1 after the clean-up step is already better than these results we can only assume our global placement algorithm has found a better initial solution than those of XQ and TimberWolf.

For primary1, biomed, primary2, industry3 and avqlarge our results are better than those produced by TimberWolf. Of course it should be noted that our run times are substantially higher.

For the golem3 circuit our result is more than 35 % worse than that of either of the two other placement heuristics. The results are even worse for the IBM-circuits which are between almost 50 and 100 % higher than those of XQ. This could be explained by the fact that XQ is geared towards large instances of standard-cells and our placement heuristic is geared towards medium sized instances of general-cells.

9.4.5 Conclusion on Final Placement

Our final placement results *are* in general good. The results are comparable with the results of commercial placers and even with the results of section 3.3 which may be for slightly different instances. Specifically we have spacing between modules (except for the ami33k circuit). This is not the case in several of the results of section 3.3.

Several other conclusions can be drawn from the test results of the previous paragraphs:

- **General-cell circuits** Our final placement heuristic is capable of placing very large general-cell instances. The final placement of the general-cell instances have netlength within 20 % of their standard-cell counterparts and two of the three are less than 15 % above their counterparts. This is definitely a success since we have succeeded in placing and improving general-cell circuits to high quality within reasonable time.

- **Legalization** The small changes of the graphs of section 9.4.3 between solutions of subsequent iterations could signify that the legalization algorithm does not miss good solutions of the final placement combinatorial search. This is a surprise considering the simple legalization criteria we imposed on the final placement algorithm

- **Large circuits** The new final placement algorithm does improve the large IBM-circuits 10 % during 9 hours but placements are still not comparable to those of XQ.

- **Speed** It should also be mentioned that our final placement is *slow*. For the medium sized test circuit less than half a million nets have been picked for relaxation based local search after three hours. For the industry2 circuit 352256 nets had been searched at the end of final placement. 103758 of those had resulted in improvements of the objective function. The major bottleneck is likely the semi-legalization step which searches a very large neighborhood.

- **Cleaning up random accepts** The fact that almost a third of the nets of the industry2 test results in improvement could signify that much of the work done by the relaxation based local search goes to cleaning up poor random accepts of the simulated annealer.

- **Ill-tuned Simulated Annealing** Many of the results and the graphs of 9.4.3 signify that the simulated annealer is far from well-tuned and may not have optimal coolage and reset strategies. These strategies were determined through preliminary tests where they worked well. Unfortunately they did not scale well with circuit size and time.

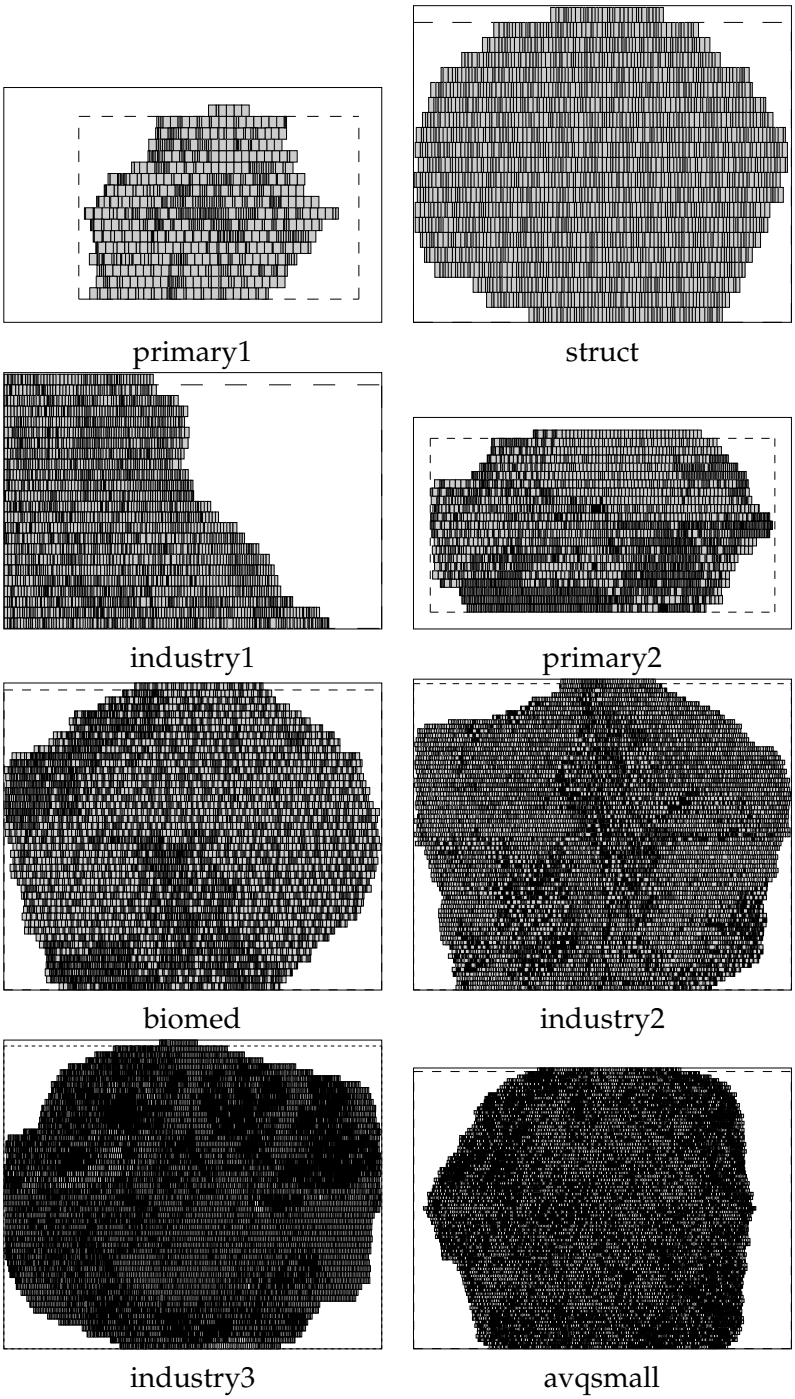
  The simulated annealer was a last minute attempt to escape the local minima we encountered otherwise. Unfortunately tests show that it is too hard to control e.g. the reset strategy. Our assumptions that the simulated annealer would function equally on small and large circuits seem over-idealized and it is not unlikely that the start time t_0 and coolage value K depends on some properties of the placement instance.

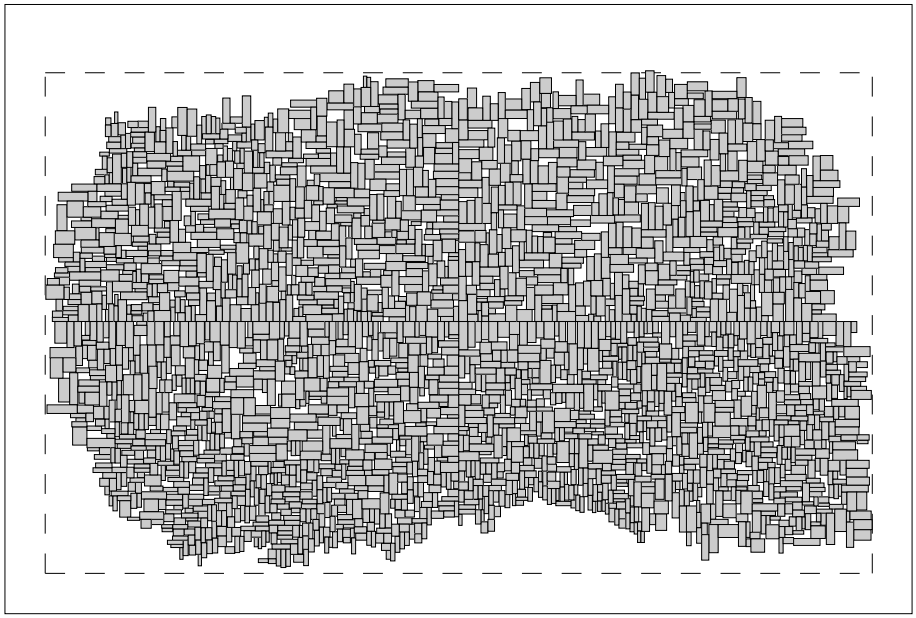
Our simulated annealer could certainly be tuned much better than the preliminary tuning of this section. One of the major problems with our tuning is that we only consider early iterations and that the values of t_0 and K are based on three small circuits. Not surprisingly it seems that the heuristic performs well on these circuits during the first 10 minutes which was exactly the run time for the tuning tests of t_0 and K.

While the simulated annealing strategy may have failed it does seem as though the relaxation based local search makes up for this and we believe that it is the main reason for the high quality placements produced after all.

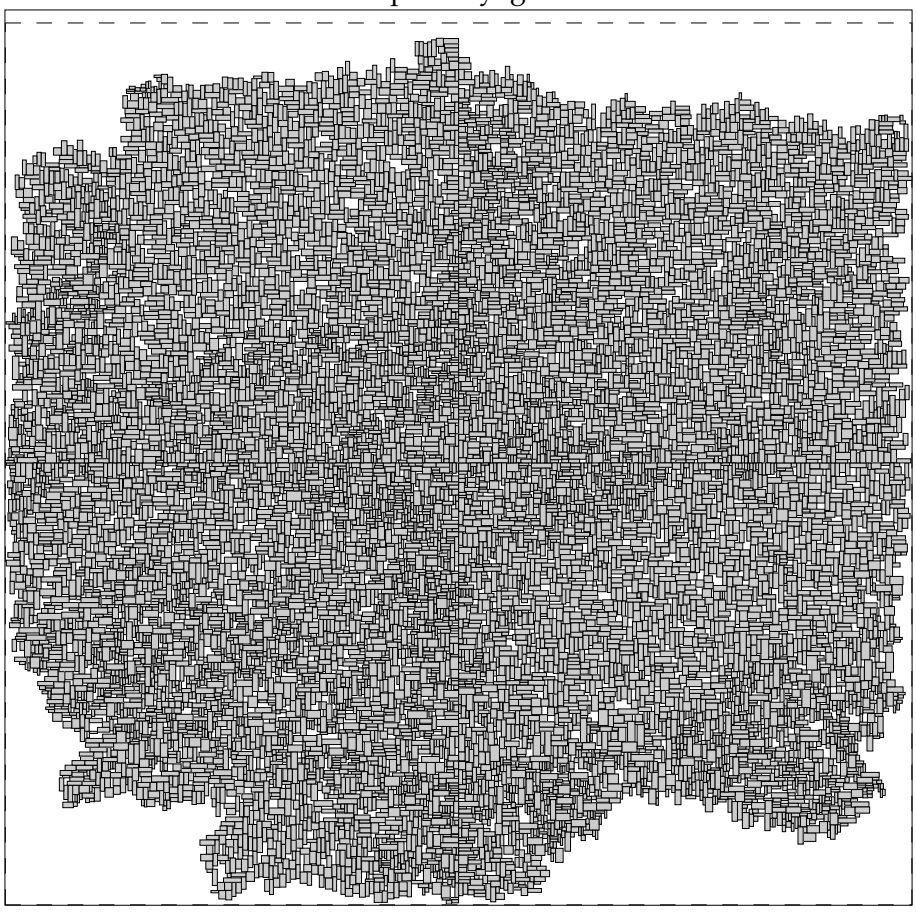
Based on the fact that this is a preliminary prototype of a new final placement heuristic the results are extremely good.

The final placement results are shown on the following pages. Please note the compactness of the circuits primary2g, industry2g and industry3g.

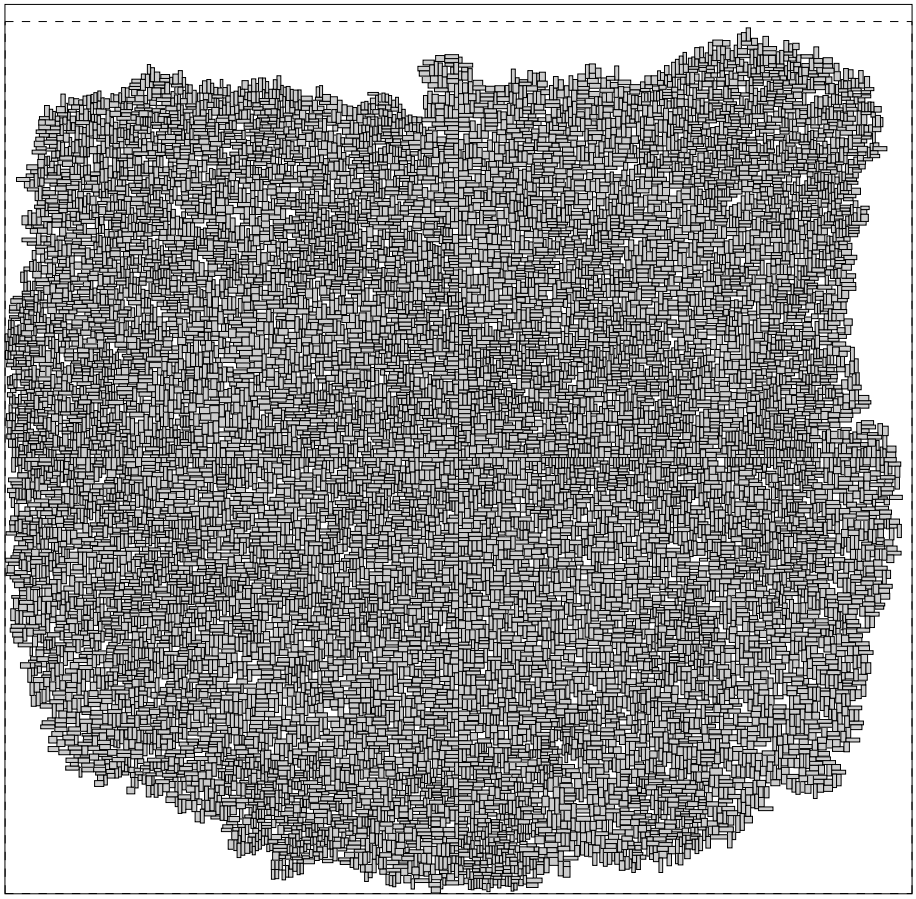

primary1
struct
industry1
primary2
biomed
industry2
industry3
avqsmall



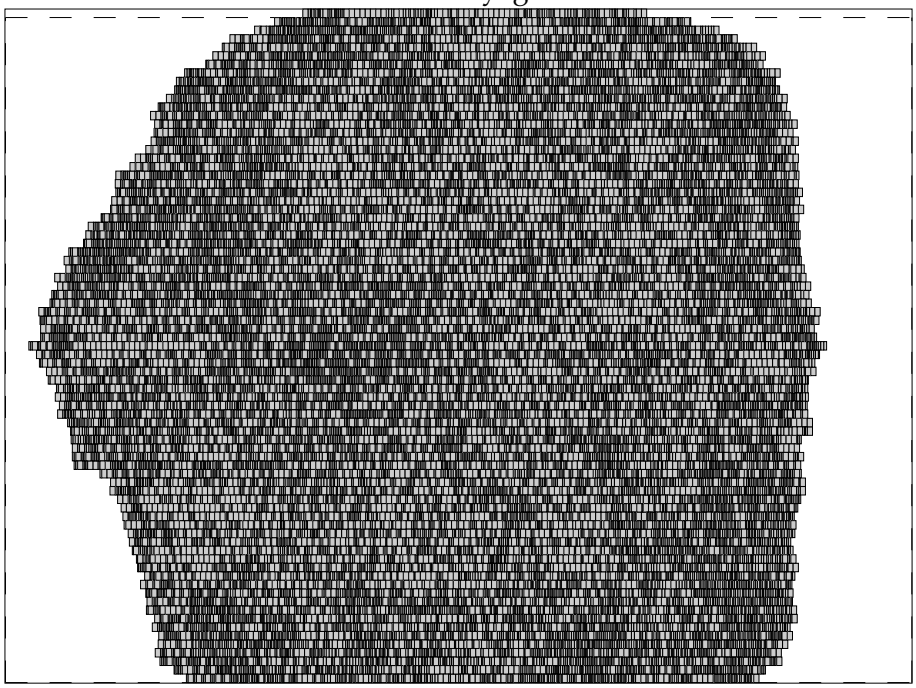
primary2g



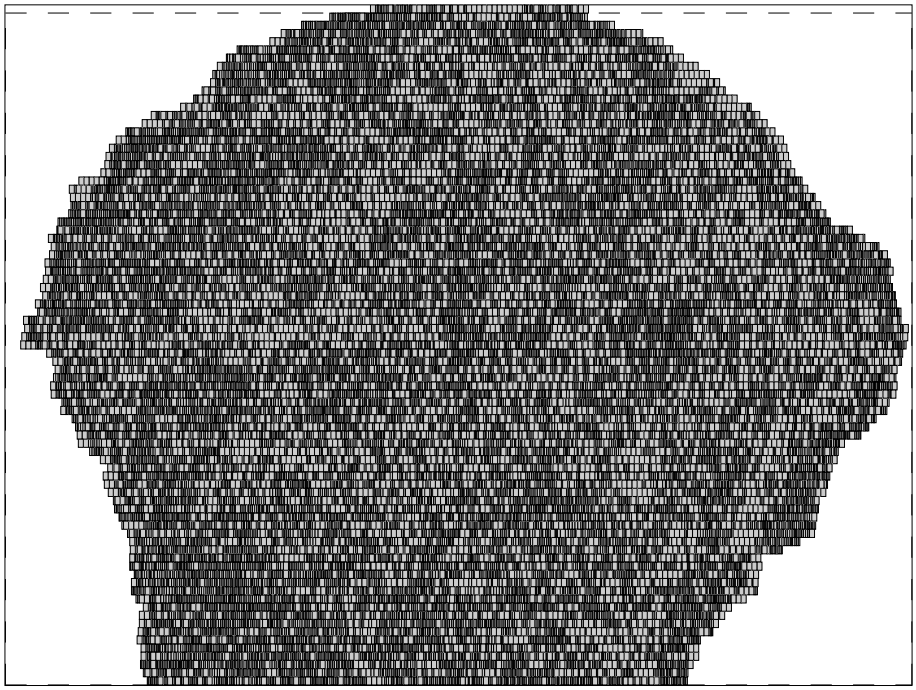
industry2g



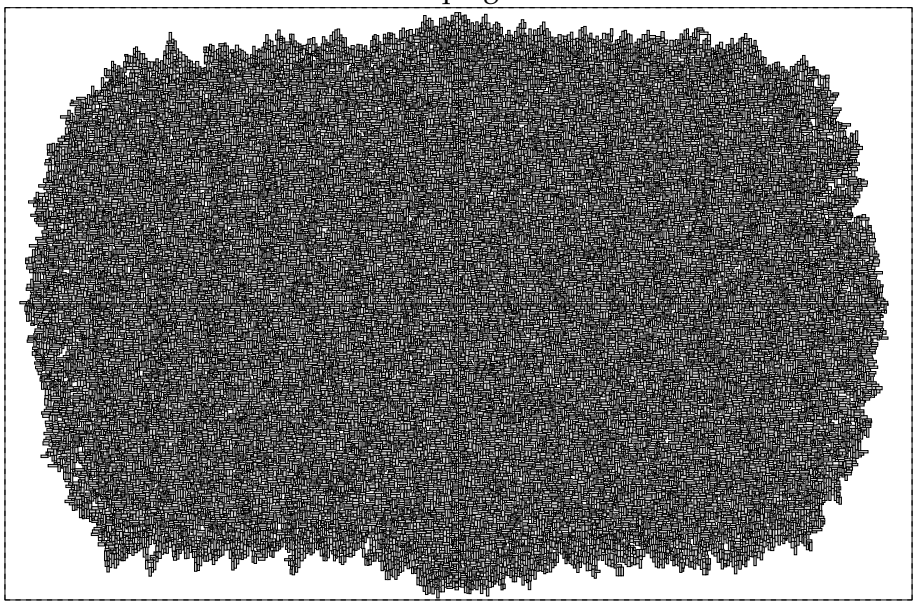
industry3g



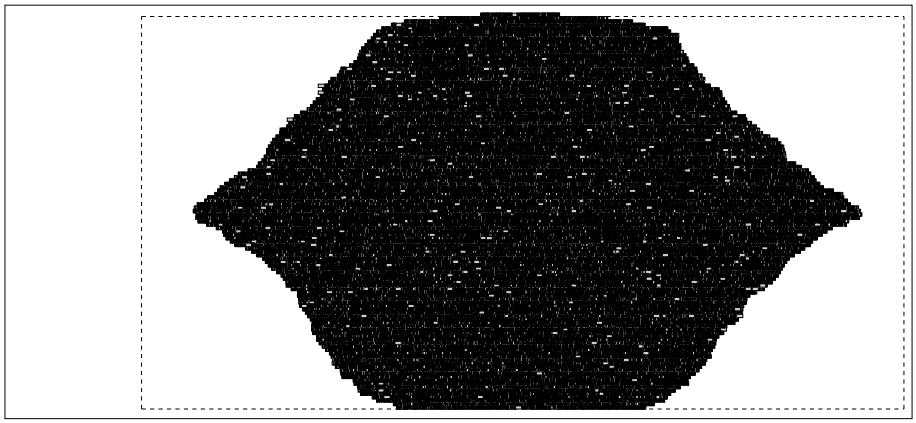
avqsmall



avqlarge

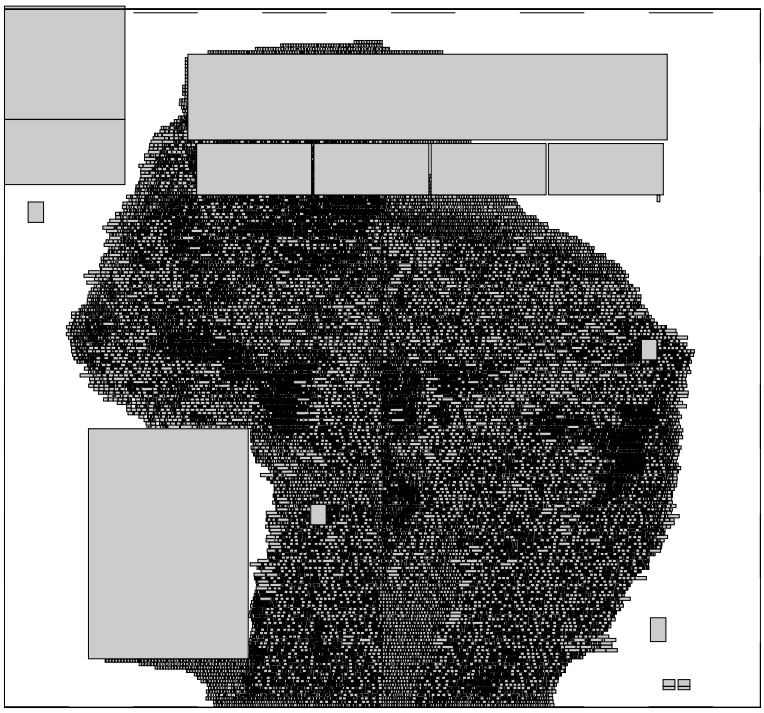


ami33k

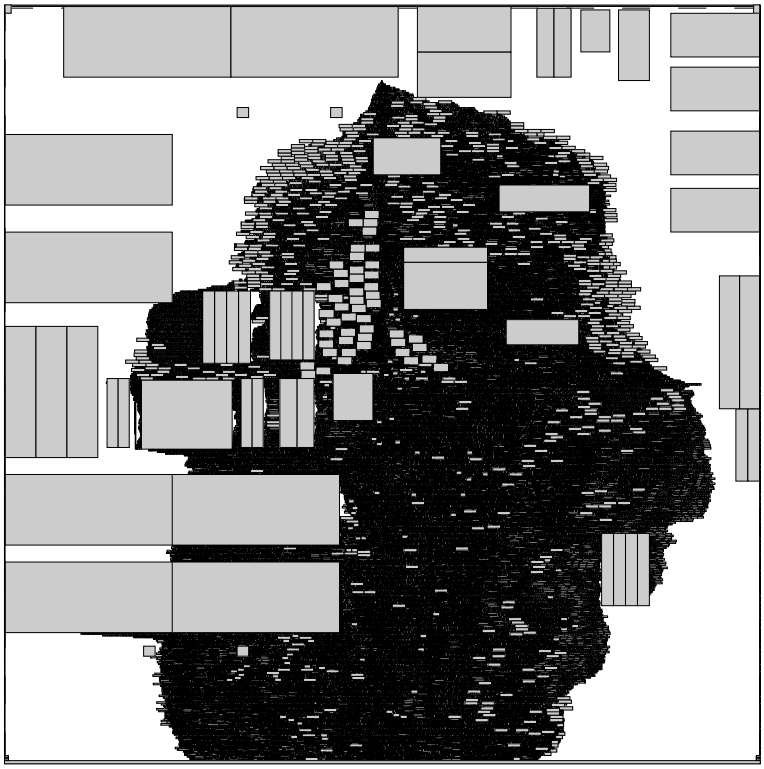


golem3

clk



decoder



pu

10 Conclusion

To summarize the previous sections have considered several topics:

- We have presented a thorough and extensive survey of placement techniques of the last decade.
- We have presented a very fast legalization algorithm which can legalize even very large circuits consisting of general-cells within seconds and we have improved an existing sequence-pair based placement method by allowing fixed modules and placement area during the placement. The new sequence-pair based placement also produce more compact placements than those of the previous sequence-pair based placement algorithms.
- We have created global placement heuristic which combined with its clean-up step produces results of small and medium sized circuits which seem comparable to those of recent global placement heuristics. This is a good result considering that our new global placement heuristic differs from previously published ideas and is only a preliminary prototype with much room for improvement.
- We have created a new final placement heuristic which is capable of improving results produced by our new global placement to produce results comparable to those of previously published and commercial placers. The new final placement is controlled by simulated annealing and the simulated annealing parameters and strategies do not seem ideal. In spite of this the placement heuristic is capable of producing quite good results which signify that the underlying ideas are good. The simulated annealer was added in last minute and was not well-tuned. A well-tuned and better control method of the relaxation based local search would probably produce even better results.

In general most of our work has been successful and we are confident that further improvements could be added to the described successful new placement techniques both in terms of speed of the heuristic and in terms of quality of solutions.

We have reached our *main goal* of being able to place general-cell circuits with high quality and it seems that we have removed at least some of the barriers between general-cells and standard-cells. With further work the new placement techniques may render general-cell placement a viable alternative to standard-cell placement over time.

A crucial subject for practical applications that we have not dealt with however is routability and congestion of our final placements. These properties could be tested with commercial routers to determine if the compact placements produced by the sequence-pair legalization can in fact be routed and used in practice.

10.1 Future Directions

Although we have covered substantial material in this thesis there is plenty of directions for future work. Our suggestions for future work is divided into the main elements of this thesis; legalization, global placement and final placement.

10.1.1 Future Work – Legalization

Future work of legalization and sequence-pair placement may consider some of the following topics.

- **Area considerations** One of the main flaws of the sequence-pair legalization technique seems to be that it does not consider area of the modules. This is likely the main reason why the simple standard-cell legalizer outperforms our sequence-pair legalizer in some cased. A new sequence-pair conversion which considers area may be able to legalize severely overlapping placements better than our current.
- **Better sliding scheme** The sliding scheme of the extended envelope based placement (see section 4.4.2) may be improved to produce even more compacted placements.
- **Improved constraint handling** The constraints imposed by VLSI-instances – limited area and fixed modules – could probably be more elegantly dealt with than we do in the extended envelope sequence-pair placer (see section 4.4.2).
- **Placement extensions** In VLSI-placement some modules could be constrained to certain regions and some modules may have rectilinear shapes. The extended envelope placement algorithm could probably be extended to handle these extra requirements.
- **3D sequence-pair** Although integrated circuits are currently two-dimensional, IBM has taken steps towards three-dimensional circuits (see [92]). Also in [28] a device nick named "gadget printer" is presented. The "gadget printer" can print electronic devices in three dimensions. Therefore new three-dimensional placement algorithms are needed. The quadratic formulations are straight-forward to extend to three dimensions, but it does seem less obvious how the sequence-pair representation should be extended to accommodate any extra dimensions.

10.1.2 Future Work – Global Placement

Our global placement algorithm is far from ideal and future work may deal with some of these topics.

- **Congestion and routability** The global placement could be extended to account for these two elements.
- **Improved artificial net scheme** The tests reveal that global placement never really converges. A new scheme which has better convergence properties should be devised.

10.1.3 Future Work – Final Placement

The new final placement heuristic is already in its preliminary state quite promising but there is *certainly* room for improvement. Future work may deal with these topics of final placement:

- **Faster semi-legalization** The main bottleneck for our final placement seems to be semi-legalization. Therefore a new form of semi-legalization may improve run times substantially.
- **Improved γ handling** Based on the experience we have gathered during our work with the new final placement algorithm better strategies for handling the overlap penalty γ could be devised. During later iterations there is little overlap penalty in the objective function. This is likely because the improving moves that exist at this stage are all very small and not sufficiently good to account for the loss in objective value due to overlap penalty.
- **Improved legalization criteria** Our simple legalization criteria – number of moves in the placement – is very crude and may miss good placements. A method which could legalize good placements at the right time during local search may improve solution quality.
- **Improved Sub-circuit Handling** For our tests we have set the sub-circuit size to 10. However it is possible that varying this size would improve the performance of final placement since in some cases the sub-circuit size may help to escape local minimum. In other cases smaller sub-circuits may be sufficient to move towards good solutions.
- **Better relaxed local search control** Although it does allow us to escape local minimum, the preliminary simulated annealing heuristic does not seem to control the relaxation based local search well. A better overall control of relaxation based local search would likely improve this. In any event the reset and coolage strategies needs rethinking. A first step towards better control would be to identify the local minima imposed by the relaxed local search.
- **Congestion and routability** An objective function based on congestion would probably make the placement heuristic more suitable for practical use.

- **Large macros** A subject which we have not discussed is optimizing large macros in the placement. This is difficult to do with the swap-based neighborhood and presents many new problems.

10.2 Epilogue

During the previous many months my experience with the VLSI-layout problem has increased manifold. It has been educational but often hard. The major breakthroughs during this thesis came with the discovery of the α-parameter of the placement-to-sequence-pair algorithms and relaxed local search with semi-legalization. Although these ideas may seem obvious it took a substantial amount of thinking and rethinking before they took the form in my mind that they have in their current state.

One of the hardest parts of the thesis was to discover the strength and weaknesses of the legalization algorithms and to come up with methods that would balance the amount of overlap in the placements.

The simulated annealer for Relaxed Local Search was implemented in very last minute as a last resort for escaping local minima. I therefore had little time to improve on it. Its present outline came from initial short-lasting tests which did not scale well with the final tests. Other and better suited simulated annealing strategies exists and could be tested in the future.

For future local search methods for final placement I believe the right direction to move in is based on relaxation based local search. It seems to be a general trend among successful approaches that some form of large scale relaxation occurs during optimization.

The VLSI-layout problem is more complex than it seems and there is certainly plenty of room for future ideas. As a rule of thumb however it appears that if you can think of a reason why your idea may not work in practice, it will not work in practice. If you can not think of a reason, there is still a good chance that it will not work anyway.

In any event I hereby encourage others to engage at this problem. There are still many unanswered questions in the field.

References

[1] C. J. Alpert, T. F. Chan, D. J. H. Huang, A. B. Kahng, I. L. Markov, P. Mulet, and K. Yan. Faster minimization of linear wirelength for global placement. In *Proceedings of 1997 International Symposium on Physical Design*, pages 4–11, 1997.

[2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration, the VLSI Journal*, 19:1–80, December 1995.

[3] S. Areibi, Matt Thompson, and Anthony Vanelli. A utility-based iterative improvement heuristic for standard cell placement. In *Proceedings of First International Conference on Engineering and Reconfigurable Systems and Algorithms*, 2001.

[4] R. Baldick, A. B. Kahng, A. A. Kennings, and I. L. Markov. Efficient optimization by modifying the objective function: Application to timing-driven vlsi layout. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(8):947–956, 2001.

[5] Florin Balsa. Symmetry within the sequence-pair representation in the context of placement for analog design. In *IEEE Transactions on CAD of IC's and Systems*, volume 19, pages 721–731, 2000.

[6] U. Brenner and J. Vygen. Faster optimal single-row placement with fixed ordering. In *Proceedings of Design, Automation and Test in Europe*, pages 117–121, 2000.

[7] U. Brenner and J. Vygen. Worst–case ratios of networks in the rectilinear plane. *Submitted to publication 2000*, 2000.

[8] J.D. Burden and R.L. Faires. *Numerical Analysis*. Brooks/Cole Publishing Company, sixth edition edition, 1997.

[9] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L . Markov, and A. Zelikovsky. On wirelength estimations for row-based placement. In *Proceedings of ACM/IEEE International Symposium on Physical Design*, pages 4–11, 1998.

[10] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Design and implementation of move-based heuristics for vlsi hypergraph partitioning. *ACM Journal of Experimental Algorithms*, 5, 2000.

[11] Y. Chung, Y. Chang, G. Wu, and S. Wu. B*-tree: A new representation for non-slicing floorplans. In *Proceedings of Design Automation Conference*, pages 458–463, 2000.

[12] F.R.K. Chunk and R.L. Graham. On steiner trees for bounded point sets. *Geom. Dedicata*, (11):353–361, 1981.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[14] A. Dasdan and C. Aykanat. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Transactions on Computer-Aided Design*, 16, Februar 1997.

[15] K. Doll, F. Johannes, and G. Sigl. Domino: Deterministic placement with hill-climbing capabilities. In *Proceedings of VLSI*, pages 3b.1.l–3b.1.10, 1991.

[16] K. Doll, F. Johannes, and G. Sigl. Accurate net models for placement improvement by network flow methods. In *Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*, pages 594 – 597, 1992.

[17] K. Dowsland. Simulated annealing. In Colin Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems.*, pages 20–63. Oxford: Blackwell, 1993.

[18] A. Dunlop, V. Agrawal, D. Deutsch, M. Juki, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *Proceedings of ACM/IEEE Design Automation Conference*, 1984.

[19] H. Eisenmann and F. M. Johannes. Generic global placement and floor planning. In *Proceedings of the 35th annual conference on Design automation conference*, pages 259–274, 1998.

[20] H. Etavil, S. Areibi, and Anthony Vannelli. Attractor-repeller approach for global placement. In *Proceedings of the 1999 IEEE/ACM international conference on computer-aided design*, pages 20 – 24, 1999.

[21] O. Faroe, D. Pisinger, and M. Zachariasen. Local search for final placement in VLSI design. In *Proceedings of ICCAD*, pages 565–572, 2001.

[22] S. Fekete and J. Schepers. On more-dimensional packing i-iii. *Koln University technical reports 97-288, 97-289, 97-290*, 1997.

[23] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of ACM/IEEE Proceedings 19th Design Automation Conference*, pages 175–181, 1982.

[24] O. Færø. Placement of modules in vlsi-layout. Master's thesis, University of Copenhagen, DIKU, 2000.

[25] M. Frederiksen. Parallel region-based vlsi placement with multiple objectives. Master's thesis, University of Copenhagen, DIKU, 2002.

[26] K. Fujiyoshi and H. Murata. Arbitrary convex and concave rectilinear block packing using sequence-pair. In *Proceedings of the 1999 international symposium on Physical design*, pages 103–110, 1999.

[27] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, (35):921–940, 1988.

[28] Duncan Graham-Rowe. "gadget printer" promises industrial revolution. *NewScientist.Com*, (http://www.newscientist.com/news/ news.jsp?id=ns99993238), 2003.

[29] L. K. Grover. A new simulated annealing algorithm for standard cell placement. In *Proceedings of International Conference on Computer-Aided Design*, pages 378–380, 1986.

[30] L. K. Grover and S. Mallela. Clustering based simulated annealing for standard cell placement. In *Proceedings of 25th Design Automation Conference*, pages 312–317, 1988.

[31] P. Guo, C. Cheng, and T. Yoshimura. An o-tree representation of non-slicing floorplan and its applications. In *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pages 268–273, 1999.

[32] L. Hagen and A. B. Kahng. A new approach to effective circuit clustering. In *Proceedings of International Conference on Computer-Aided Design*, pages 422–427, 1992.

[33] L. W. Hagen, D. J. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning methods. *Proc. European Design Automation Conference*, pages 144–149, 1995.

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

[35] M. R. Hestenes and E. Stiefel. Methods of conjugate gradient for solving linear systems. *Journal of Research of the National Bureau of Standards*, pages 409 – 436, 1952.

[36] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, and C. Cheng. Corner block list: An effective and topological representation of non-slicing floorplan. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 8 – 12, 2000.

[37] B. Hu and M. Marek-Sadowska. Far: Fixed point addition and relaxtion based placement. In *Proceedings of 2002 International Symposium on Physical Design*, pages 161–166, 2002.

[38] S. Hur and J. Lillis. Relaxation and clustering in a local search framework: Application to linear placement. In *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pages 360–366, 1999.

[39] S. Hur and John Lillis. Mongrel: Hybrid techniques for standard cell placement. In *Proceedings of the International Conference on Computer Aided Design*, pages 165 – 170, 2000.

[40] F. K. Hwang. On steiner minimal trees with rectilinear distance. *Siam Journal of Applied Mathematics*, (January):104–114, 1976.

[41] S. Imahori, M. Yagiura, and T. Ibaraki. Local seach algorithms for the rectangle packing problem with general spatial cost. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 63–82. Kluwer Academic Publishers, Boston, 1996.

[42] M. Jackson and E. Kuh. In *Proceedings of ACM/IEEE Design Automation Conference*, volume 26, pages 370 – 375, 1989.

[43] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part 1, graph partitioning. *Operations Research*, 37:865–892, 1989.

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

[45] M. Z. Kang and W. W. Dai. Arbitrary rectilinear block packing based on sequence pair. In *Proceedings of the 1998 IEEE/ACM international conference on CAD*, pages 259–266, 1998.

[46] A. A. Kennings and I. L. Markov. Analytic minimization of half perimeter wirelength. In *Proceedings of the 2000 Conference on Asia and South Pacific Design Automation*, pages 179–184, 2000.

[47] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[48] S. Kirkpatrick, C Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, (4598):671–680, 1983.

[49] J. M. Kleinhans, G. Sigl F. M. Johannes, and K. J. Antreich. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided-Design of Integrated Circuits and Systems*, (10):356–365, 1991.

[50] K. Kozminski. Benchmarks for layout synthesis - evolution and current status. In *proceedings of ACM/IEEE Design Automation Conference*, pages 265–270, 1991.

[51] C. Kring and K. Newton. A cell-replicating approach to mincut-based circuit partitioning. *IEEE Intl. Conf. on Computer Aided Design*, pages 2–5, 1991.

[52] B. Krishnamurthy. An improved min-cut algorithm for partitioning vlsi networks. *IEEE Transactions on Computing*, 33:438–446, May 1984.

[53] J. Lai, M. Lin, T. Wang, and L. Wang. Module placement with boundary constraints using the sequence-pair representation. In *Proceedings of the conference on Asia South Pacific Design Automation Conference*, pages 515–520.

[54] T. Lengauer. *Combinatorial Algorithms for Integrated Circuits.* Wiley-Teubner, 1990.

[55] C. Lin. A more efficient sequence pair pertubation scheme. In *IEEE Proceedings RISC 99*, pages 295–300, 1999.

[56] J. Lin and Y. Chang. Tcg: A transitive closure graph-based representation for non-slicing floorplans. In *Proceedings of Design Automation Conference*, pages 764 – 769, 2001.

[57] P. Madden. Reporting of standard cell placement results. *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, 21(2):240 – 247, 2002.

[58] I. I. Mahmoud, K. Asakura, T. Nishibu, and T. Ohtsuki. Experimental appraisal of linear and quadratic objective functions effect on force direction method for analog placement. In *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, number 4, pages 710–725, 1994.

[59] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, (48):256–267, 2000.

[60] J. McClellan, R. Schafer, and M. Yoder. *DSP First - A Multimedia Approach*. Prentice Hall, 1999.

[61] F. Mo, A. Tabbara, and R. K. Brayton. A force-directed macro-cell placer. In *Proceedings of International Conference on Computer-Aided Design*, number 4A.3, 2000.

[62] H. Murata, K. Fujiyoshi, and M. Kaneko. Vlsi/pcb placement with obstacles based on sequence pair. In *Proceedings of the 1997 international symposium*, pages 27–31, 1997.

[63] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. In *IEEE Transactions on CAD*, volume 15, pages 1518–1524, 1996.

[64] H. Murata and E. Kuh. Sequence-pair based placement method for hard/soft/pre-placed modules. In *Proceedings of the 1998 international symposium on Physical design*, pages 167–172, 1998.

[65] Sudip Nag and Kamal Chaudhary. Post-placement residual-overlap removal with minimal movement. In *Proceedings of Design Automation and Test in Europe*, pages 581–587, 1999.

[66] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on bsg-structure and ic layout applications. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 484–491, 1996.

[67] S. Olsen. *Forelæsningsnoter til Billedbehandling*. http://www.diku.dk/forskning/image/teaching/Courses/e99.101/, 1999.

[68] H. Onodera, K. Fujiyoshi, and K. Tamaru. Branch and bound placement for building block layout. In *Proceedings of the 28th conference on ACM/IEEE desing automation conference*, pages 433–439, 1991.

[69] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, (41):338–350, 1993.

[70] Y. Pang, C. Cheng, K. Lampaert, and W. Xie. Rectilinear block packing using o-tree representation. In *Proceedings of the 2001 international symposium on Physical design*, pages 156 – 161, 2001.

[71] P. Parakh, R. Brown, and K. Sakallah. Congestion driven quadratic placement. In *Proceedings of the 35th annual conference on Design automation conference*, pages 275 – 278, 1998.

[72] D. Pisinger. From sequence pair to semi-normalized placement in $o(n \log \log n)$ time. *Submitted ICCAD 2002*, 2002.

[73] Ken Popovich. Intel looks to 1 billion-transistor chip. *eWeek*, (http://www.eweek.com/article2/0,3959,636064,00.asp), 2002.

[74] J. K. Reid. On the methods of conjugate gradient for the solution of large sparse systems of linear equations. In *Large Sparse Sets of Linear Equations*, pages 231–254. Academic Press, 1971.

[75] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

[76] M. Sarrafzadeg and M. Wang. Nrg: Global and detailed placement. In *Proceedings of International conference on Computer-Aided Design*, pages 532 – 537, 1997.

[77] J. Schewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, CS, Carnegie Mellon University, http://www.cs.cmu.edu/ jrs/jrspapers.html, 1994.

[78] C. Sechen and K. W. Lee. An improved simulated annealing algorithm for row-based placement. In *Proceedings of International Conference on Computer-Aided Design.*, pages 478–481, 1987.

[79] G. Sigl, K. Doll, and F. M. Johannes. Analytical placement: A linear or a quadratic objective function. In *Proceedings of 28th ACM/IEEE Design Automation Conference*, pages 427–432, 1991.

[80] W. Sun and Carl Sechen. Efficient and effective placement for very large circuits. In *Proceedings of International Conference on Computer Aided Design*, pages 170–177, 1993.

[81] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of the conference on Design, automation and test in Europe*, pages 106–111, 2000.

[82] X. Tang and D. F. Wong. Fast-sp: a fast algorithm for block placement based on sequence pair. In *Proceedings of the conference on Asia South Pacific Design Automation Conference*, pages 521–526, 2001.

[83] R. Tsay, E. S. Kuh, and C. Hsu. Proud: A fast sea of gates placement algorithm. In *Proceedings of the 26th ACM/IEEE Conference on Design Automation*, pages 318 – 323, 1988.

[84] P. van Emde Boas, R. Kaas, and E. Zulstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10(2), 1977.

[85] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, (113):469–499, 1999.

[86] J. Vygen. Algorithms for large-scale flat placement. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 746–751, 1997.

[87] J. Vygen. Plazierung im vlsi-design und ein zweidimensionales zerlegungsproblem, 1997.

[88] J. Vygen. Algorithms for detailed placement of standard cells. In *Proceedings of Design, Automation and Test in Europe 1998*, pages 321 – 324, 1998.

[89] M. Wang and X. Yang M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proceedings of International Conference on Computer Aided Design*, pages 260 – 263, 2000.

[90] D. Warme, P. Winter, and M. Zachariasen. Exact algorithms for plane steiner tree problems: A computational study. In Ding-Zhu Du, editor, *Advances in Steiner Trees*, pages 81–116. Kluwer Academic Publishers, 2000.

[91] B. X. Weiss and D. A. Mlynsky. A graphtheoretic approach to the relative placement problem. In *Proceedings of IEEE Transactions on Circuits and Systems*, pages 286–293, 1988.

[92] (WWW). Ibm creates new dimension for high-performance chips. *IBM Research News*, (http://www.research.ibm.com/resources/news/ 20021111_3d_ic.shtml), 2002.

[93] J. Xu, P. Guo, and C. Cheng. Rectilinear block placement using sequence-pair. In *Proceedings of the 1998 international symposium on Physical design*, pages 173–178, 1998.

[94] C. W. Yeh, C. K. Cheng, and T. T. Lin. A probabilistic multicommodity-flow solution to circuit clustering problems. In *Proceedings of International Conference on Computer-Aided Design*, pages 428–431, 1992.

A Unsuccessful approaches

This section is devoted to people who intend to create their own heuristic for solving the placement problem. It describes some of the many methods we tried during the writing of this thesis which did not work well. Although negative results are not commonly a part of a scientific text we have included them so that others may avoid wasting the time we did. Also the section may serve as an inspiration. What did not work out for us may work for others with simple modifications. In most cases we have included what we believe is the explanation for the poor results.

The approaches have been grouped in three main parts; sequence-pair related approaches, global placement approaches and local search approaches.

It should be noted that methods presented below are sketchy but in most cases various modifications and enhancements as well as parameter adjustments were attempted without any luck. Should you find inspiration in the mentioned methods we wish you the best of luck.

A.1 Unsuccessful Sequence-Pair Conversions

Various methods were attempted at converting the sequence-pair placement to make it handle uneven distributions of modules better by interpreting placements differently:

- **Polar coordinates** One method used polar coordinates The modules ordered by distance from origin was interpreted as the $\mathbf{B}$-sequence and the angle as the A-sequence.
- **Percentile distance** Another method using percentile distance along the $\mathbf{B}$-diagonals of algorithm 4.1 instead of absolute coordinates to determine the $\mathbf{A}$-sequence.
- **Partition based sequence-pair** A partition based placement-to-sequence-pair converter which could be used for placements with much overlap was also examined. The idea was to bi-sect as a common partitioner but the constructed floor-plan was put into a sequence-pair since this is relatively easy to do. The purpose of this attempt was to account for the area of modules.
- **Area based** Here modules were given two values depending on how much area of other modules was below and left of them and above and left of them. The modules were then sorted according to these two values to give respectively the $\mathbf{A}$- and $\mathbf{B}$-sequences.
- **Connection based sequence-pair transform** We also tried to use the connection graph for conversion. By using diagonals one of four relations – left, right,

above, lower – was retrieved for any two connected modules. A new graph was constructed in which each node corresponded to a module or an IO-pin and four types of edges (left, right, above, lower) were added to this graph between connected components of the connection-graph of the circuit depending on their internal relation. The graph was then converted to a sequence-pair using topological sorting (see e.g. [13]) based on the edge-types, such that the top-right module was last in the $\mathbf{A}$-sequence and the lower-right module last in the $\mathbf{B}$ sequence. This can be done by doing a topological sort on up/right edges for the $\mathbf{A}$-sequence and lower-right edges for the $\mathbf{B}$ sequence. In other words only connected components were used when constructing the sequence-pair.

In all cases the results were poor – presumably because there is too much inconsistency between capture of the placements and repositioning. The area based method did show some potential but did not handle modules that were close to each other well. The connection method was able to determine connected components and place them close to each other. Unfortunately it was unable to detect the *right* clusters to be placed next to each other. Modules which were placed far from each other but connected could end up next to each other and drag other modules with them in an obvious wrong direction.

One of the main problems of the placement-to-sequence-pair algorithm is that modules are often distributed unevenly on the placement area. Therefore we tried several module-spreading methods:

- **Partitioning for spreading** One method was based on a recursive bi-partitioning scheme. In each step the partitions were assigned modules such the sum of the area of their modules was roughly half. The recursion stopped when a region contained k modules. When the entire bi-partitioning was completed the modules were redistributed in each section of the real-estate described by the bi-section.

- **Dense area warping** A second method was based on thin-plate-spline warping functions. A warping function was constructed such that areas with many modules were spread out and areas with few modules were reduced under warping. Unfortunately the warping method was far too difficult to control for it to work in practice.

- **Triangular spring system** A third method used delaunay-triangulation to setup connections between modules close to each other in the current placement. Assuming that the distances between modules in general should be equal a auxiliary "spring-system" similar to that of the quadratic nets was constructed with springs between nearby modules. The spring-system was put into equilibrium in the hope that the distances between modules would even out.

In all three cases the main problem was that the spreading of the modules occurred independently of the objective function and that the relative positions which are implicitly stored in the overlapping placement was also lost after spreading. In all three cases the spread placements lead to worse results than no spreading with an appropriate α-value. However if the standard placement-to-sequence-pair algorithm was not allowed to α-search, spreadings could some times produce better results.

A.2 Unsuccessful Global Placement Improvement Strategies

- **Relaxation based method** The first attempt at improving a legal solution was to relax the overlapping constraints for a part of the circuit. A specific amount of modules, say 10%, were moved to optimal positions without considering overlap. The resulting placement was then legalized. The results were poor presumably because legalization of the resulting massively overlapping placement proved to difficult for our simple legalizer. Adjustments of the amount of modules to move did not achieve anything. Small percentages resulted in poor legalizations without an equal improvement in placement quality since few modules were improved. Large percentages made the legalization too difficulty. Over time the legalized placement began to differ substantially from the initial placement and all information from it was lost. The end result was actually increasingly worse legalized solutions. We also tried variants in which the modules were slowly dragged to their optimal positions but the results were quite similar.

- **Force-like method** We attempted a force like method with increasing net-weights. The method started with a legal placement and a quadratic program was set so that the legal placement would be solution to the *relaxed* problem of the quadratic program. At this point net-weights were increased for the longest nets until legalization of the relaxed problem differed from the initial legalized placement. At this time net-weights were reset and the heuristic restarted. The problem with this method seemed to be that it was too hard to control. Large increases in weight altered the solution too much. Small increases made the method too slow. Also increasing weight of long nets seemed only to increase length of the shorter but less weighted nets thus increasing the total weight.

- **Locking of closest modules** Another method was to lock modules which moved the least during the legalization step so that their position would be retained in subsequent quadratic optimizations. The method would lock e.g. 10% of all modules in each iteration. This did not work well. In general modules which were positioned first during the sequence-pair placement step were locked. These are often closest to the center of the gravity where most modules end up during quadratic optimization. This is where there is most overlap and therefore the

legalizer performs the worst retaining little information from the quadratic solution. Also this did not spread out modules as was desired. Rather modules from subsequent quadratic optimizations were positioned on top of the locked modules making legalization impossible without moving the locked modules. Since moving the locked modules made the solution differ more from the initial solution the result was increasingly bad solutions. Attempts with unlocking strategies (e.g. 5% worst positioned of the locked) did not improve the situation notably.

- **Net penalizing during global placement** We tried various forms of increasing weights of long nets (based on bounding box netlength) during global placement so that they would decrease in length in the next iteration of the quadratic placer. The biggest problem with this method is likely that the quadratic objective function is sufficiently close to the bounding-box formulations and altering weights will have the opposite effect; increase length of connected nets.

- **Loosening of nets** We tried to loosen nets during initial iterations to spread the modules more. Preliminary tests convinced us that the method dealt too little with actual non-overlapping placements and in general only increased net lengths.

A.3 Unsuccessful Final Placement Techniques

- **Relaxation cycles** The initial approach for relaxation was similar to the final method; to extract a subset of the modules and solving the unconstrained placement problem for these. Rather than simply placing them at their relaxed position we required that the modules interchanged positions. Specifically: 1) The relaxed placement was constructed. 2) A set of candidate new positions were constructed from the old positions of the relaxed modules. 3) For each module the closest of the candidate positions to its relaxed position was chosen (much like semi-legalization). If the new placement resulted in an improvement in netlength it was accepted. Otherwise rejected.

  To increase the search space part permutations were considered. Any permutation consist of a *set* of sequences of the form $a_1 \rightarrow a_2, \ldots, a_k \rightarrow a_1$. Such a sequence may be considered a cycle of moves since a_k moves back to where a_1 was. Now the permutation arising from placing the relaxed modules was searched for such cycles and each cycle was placed individually. If any cycle improved the netlength it was accepted. The biggest problem with this method seemed to be that of extracting suited sub circuits.

  We tried both to extract strongly connected modules and modules which lay close to each other. Neither method worked very well. In general only few cycles resulted in an improvement. Presumably because the candidate positions were

far from the relaxed placements or modules piled together on top of each other. One might attempt to improve the semi-legalization by constructing an assignment problem which can be used to minimize total movement from relaxed to semi-legalized position. We rejected this method because solution methods for assignment problems have cubic running times. However for small sub circuits this may not be a problem.

- **k-moves** Another interesting idea was the k-move idea. A simple local search heuristic may swap two modules if the swap improves the total netlength and otherwise reject it. In heuristics for other problems e.g. the traveling salesman problem and to some extend graph partitioning a deeper search space improves the heuristics. Since simple swapping can improve a global solution by say 10-15 % one might assume that adding moves in which k modules or more are moved in one step may improve the solution by a few percent.

  The method was implemented such that whenever the local search heuristic could not improve the solution by swapping two modules higher order moves where considered. The moves were conducted such that if no improving swaps for module a could be found the best n_0 swaps $a \leftrightarrow b_1, \ldots, a \leftrightarrow b_{n_0}$ were selected. Now it was checked whether a module c could be found such that moves: $a \rightarrow b_i$, $b_i \rightarrow c$, $c \rightarrow a$ would improve the netlength for various $1 \leq i \leq n_0$. If no such module combination b_i, c was found the n_1 best c's were chose and the heuristic now attempted to find a d such that $a \rightarrow b_i$, $b_i \rightarrow c_j$, $c_j \rightarrow d$, $d \rightarrow a$ for $1 \leq i \leq n_0$, $1 \leq j \leq n_1$. The procedure could continue like this for up to k long sequences. The n's were chosen decreasingly.

  Unfortunately including the k-moves hardly improves the solution quality. Attempts with 3-5- moves improved the solutions by less than 1 percent compared to the simple swapping heuristic. The price was a *substantially* increased computational time. Even if the run time could be reduced it does not seem profitable to go down this path. The biggest problem is that it was hard to find improving moves of 3 or more modules and in most cases the improvement of the higher order moves was also *very* small.

- **Guided local search variations** Several different approaches based on guided local search were attempted. Firstly the guided local search based placement technique of [24] without overlapping features and with a swap-neighborhood was attempted. Unfortunately it appears that the reason the method was able to function well was because no-overlap constraints were relaxed during local search.

  Another approach was geared towards the relaxation based local search. Here we tried to give nets features if they violated some estimate of minimum size for the nets. This did not work well. Presumably because good solutions can easily contain nets with cover a large area and because we did have an easy way to

reduce size of nets with swap-based neighborhood.

- γ**-adjustments** We also tried to adjust γ during relaxed local search. Whenever a local minimum was detected – if only few improving moves could be conducted – γ was lowered slightly. On the other hand if may improving moves could be conducted γ was raised slightly. The problem with strategy is that there seem to be a lower limit for acceptable values of γ. If γ falls below this limit placements will have too much overlap.

- **Local minima escape by relaxation** Another strategy was to use the large neighborhood solely for escaping local minima imposed by the swap-based local search. Unfortunately the problem with this strategy is that the relaxation based local search works best if the solution is not at a complete local minimum with respect to the swap-based neighborhood.

B Comparison of Standard and Extended Semi-Normalized Compaction

In this section we investigate if the extended envelope can compact general-cell circuits faster than the standard semi-normalized placement algorithm for sequence-pair. In order to test this both the standard and the extended envelope sequence-pair method were implemented in a simulated annealing framework as shown in algorithm B.1.

Algorithm B.1: Simulated annealing for compaction

Choose initial random sequence-pair (A, B);
$BestSolution =$ DoPlacement(A, B);
$T = StartT$;
$LastSolution = BestSolution$;
while $Time < MaxTime$ **do**
  Move(A, B) ;
  $CurrentSolution =$ DoPlacement(A, B);
  $Accept =$false ;
  if $CurrentSolution < BestSolution$ **then**
    $CurrentSolution = BestSolution$;
    $Accept =$true;
  else
    $s =$ [Random number from $[0, 1]$];
    $delta = (CurrentSolution - LastSolution)/LastSolution$;
    p = $e^{-\frac{delta \cdot T}{K}}$;
    if $s < p$ **then**
      $Accept =$true;
  if *Accept* **then**
    T = T + 1;
  else
    UndoMove(A, B)

In order to determine the optimal constant K a range of tests were made on each of the five small MCNC-benchmarks.

Values of K are chosen from the set:

$$S = \{5, 10, 15, 20, 25, 30, 40, 60, 75, 100, 125, 150, 175, 200, 300, 400, 500\}$$

For each value of K 25 different runs were made on each circuit lasting $10, 15, 15, 30$ and 60 seconds for respectively apte, xerox, hp, ami33 and ami49. The average result for each K is compared with the best run of each circuit and is this is shown on figure B.1.

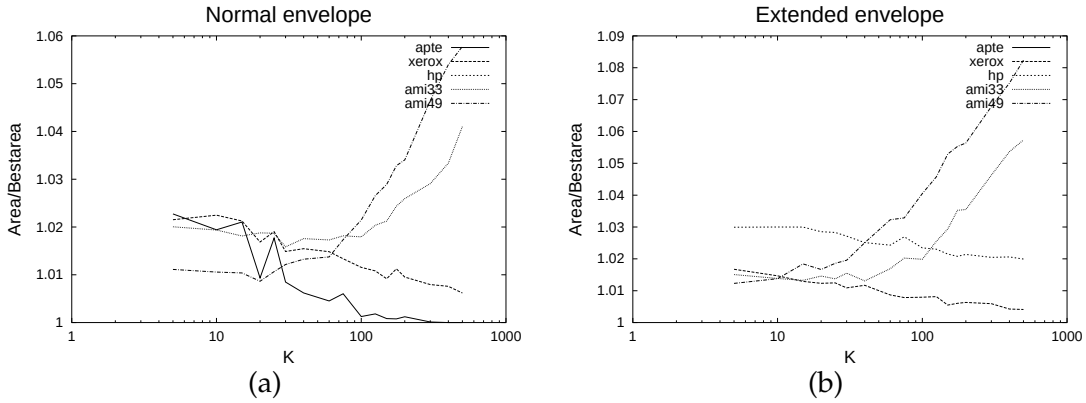


Figure B.1: *Comparison of the standard and the extended sequence envelope placement algorithm for compaction of small general-cell circuits. (A) Standard envelope. (B) Extended envelope. For a variety of test values for K 25 different runs were made. The average result of the 25 runs is plotted for each value of K. The average resulting area is divided with the total best achieved.*

Circuit	Standard - Area(Sec.) / K	Extended - Area(Sec.) / K
apte	47.07 (0) / 500	46.92 (0.02) / 500
xerox	19.80 (0.01)/ 100	19.80 (0.07) / 150
hp	8.947 (0.03)/ 400	8.847 (0.03) / 500
ami33	1.168 (5.75)/ 10	1.181 (3.61) / 40
ami49	36.16 (44.4)/ 20	36.23 (42.4) / 5

Table B.1: Best results from all runs on both the standard and extended envelope methods. Note that these results are also the best reported (see table 3.2).

B.1 Area Compaction Results

The best results for each circuit is displayed in table B.1. Note that quality, time and value of K is close to equal for the two methods. In order to compare the two methods we have shown how the area develops in the best run of each circuit both extended and normal envelope figure B.2. In general it would seem that the extended method has a slightly steeper descent, but there is a longer period between improvements than for the standard method. This could be explained by the fact that measurements have shown that decoding a sequence-pair takes almost four times longer with extended-envelope method than the normal method.

Conclusion on results In general the extended envelope does not improve the compaction over the standard envelope. This can be explained by the fact that the extended envelope uses almost four times as much time to decode a sequence-pair as

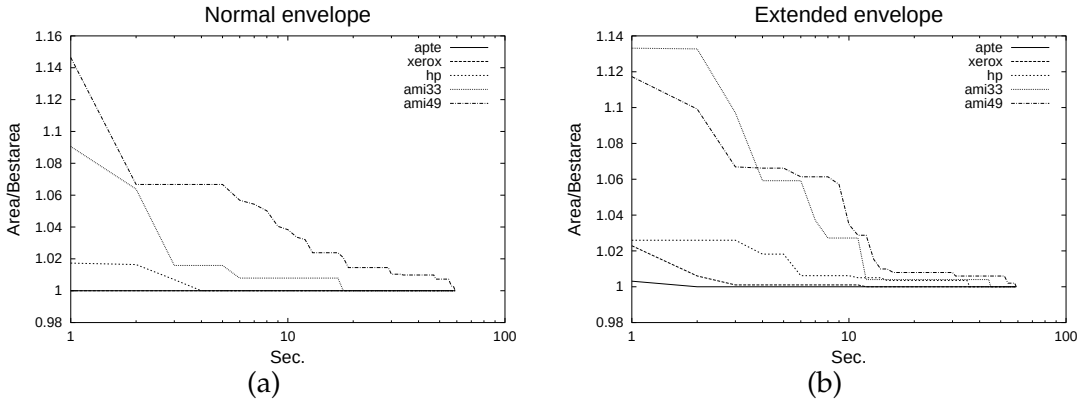


Figure B.2: *Comparison of best run of the standard and the extended sequence envelope placement algorithm for compaction of small general-cell circuits. The graph shows how the area develops during the best run of each circuit of 60 seconds. (A) Standard envelope. (B) Extended envelope.*

the standard method. So while the methods share asymptotic running time the constants of the extended envelope are larger because of the extra compaction considerations involved. Also the sequence-pairs of the later steps of the simulated annealing compaction may already be very compact and no extra gain comes from the extended envelope.

We therefore conclude that the new method may in general produce more compact placements than the standard method, but for simulated annealing driven compaction both methods perform equally well.

C Formulations for Unconstrained Optimization

In this section we present the details on how to formulate the unconstrained placement problems based on quadratic clique and star netlengths. and linear bounding box netlengths.

C.1 Matrix Formulations of Quadratic Netlengths

Both of the two popular quadratic netlength formulations; clique and star can be rewritten with matrix formulation. The following lemma will simplify the subsequent discussion.

Lemma C.1. *The quadratic clique netlength and the quadratic star netlength may be expressed as a sum of two independent functions of respectively the x- and y-coordinates of absolute coordinates of the pins.*

Proof. This easily proven since we have:

$$\begin{aligned} CL_2(n) &= \frac{1}{2|n|-2}\sum_{p\in n}\sum_{q\in n}\Big((x(c(p))+\mathrm{ofs}_x(c(p))-x(c(q))-\mathrm{ofs}_x(q))^2 \\ &\quad +(y(c(p))+\mathrm{ofs}_y(c(p))-y(c(q))-\mathrm{ofs}_y(q))^2\Big) \\ &= \frac{1}{2|n|-2}\sum_{p\in n}\sum_{q\in n}(x(c(p))+\mathrm{ofs}_x(c(p))-x(c(q))-\mathrm{ofs}_x(q))^2 \\ &\quad +\frac{1}{2|n|-2}\sum_{p\in n}\sum_{q\in n}(y(c(p))+\mathrm{ofs}_y(c(p))-y(c(q))-\mathrm{ofs}_y(q))^2 \\ &= CL_{2x}(n)+CL_{2y}(n), \end{aligned} \tag{C.1}$$

with $CL_{2x}(n)$ and $CL_{2y}(n)$ defined appropriately, similarly we can write

$$ST_2(n) = ST_{2x}(n)+ST_{2y}(n), \tag{C.2}$$

with

$$ST_{2x}(n) = \sum_{p\in n}(x(c(p))+\mathrm{ofs}_x(p)-st_{2x}(n))^2. \tag{C.3}$$

and $ST_{2y}(n)$ defined similarly where $ST_{2x}(n)$ is independent of the function y and $ST_{2y}(n)$ is independent of x. □

Since the functions in respectively x- and y-coordinate are completely symmetric and minimizing their sum is equivalent to minimizing each function separately we will only discuss minimization of the x-component functions in the following.

Again we need a little auxiliary notation. In the following theorem we let

$$\widetilde{w}(n) = \frac{w(n)}{2|n^p| - 2} \tag{C.4}$$

Theorem C.1. *If $M : \mathcal{M} \to \{1, 2, \ldots, s\}$ is a map of the s free modules of a circuit then the x-coordinate of each module can be expressed as a vector $\tilde{\mathbf{x}} \in \mathbb{R}^s$ and the x-component of the total* clique *netlength of a placement can be expressed using matrix-notation as:*

$$\sum_{n \in \mathcal{N}} CL_{2x}(n) \quad = \quad \tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} + \mathbf{d}^t \tilde{\mathbf{x}} + f \tag{C.5}$$

where $\mathbf{C}$ is an $s \times s$ matrix, $d \in \mathbb{R}^s$ and $f \in \mathbb{R}$ are defined as follows:

$$\begin{aligned}
c_{ij} &= \begin{cases} 2\sum_{i=1}^{s} \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=i}} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}_(n) & \text{if} \quad i = j \\ -2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ M(c(p))=i}} \sum_{\substack{q \in n \\ M(c(q))=j}} \widetilde{w}(n) & if \quad i \neq j \end{cases} \\
d_i &= 4 \sum_{n \in N} \sum_{\substack{p \in n \\ M(c(p))=i}} \widetilde{w}(n) \Big(\sum_{\substack{q \in n \\ q \neq p}} (\mathrm{ofs}_x(p) - \mathrm{ofs}_x(q)) + \sum_{\substack{q \in n \\ c(q) \in \mathcal{F}}} x(c(q)) \Big) \\
f &= \sum_{n \in \mathcal{N}} \sum_{p \in n} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}(n) \Big(\mathrm{ofs}_x(p)^2 + \mathrm{ofs}_x(q)^2 - 2\mathrm{ofs}_x(p)\mathrm{ofs}_x(q) \Big) + \\
&\quad 4 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} x(c(p)) \widetilde{w}(n) \Big((\mathrm{ofs}_x(p) - \mathrm{ofs}_x(q)) \Big) - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} \sum_{\substack{q \in n \\ c(q) \in \mathcal{F} \\ q \neq p}} \widetilde{w}(n) x(c(q)) x(c(p)) + \\
&\quad 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}(n) x(c(p))^2
\end{aligned} \tag{C.6}$$

Proof. Expanding the matrix-formulation of (C.5) we get:

$$
\begin{aligned}
& \tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} + \mathbf{d}^t \tilde{\mathbf{x}} + f \\
= \; & \sum_{i=1}^{s} \sum_{j=1}^{s} \tilde{x}_i \tilde{x}_j c_{ij} + d\tilde{x} + f \\
= \; & 2 \sum_{i=1}^{s} \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=i}} \sum_{\substack{q \in n \\ q \neq p}} \tilde{x}_i^2 \widetilde{w}(n) - 2 \sum_{i=1}^{s} \sum_{\substack{j=1 \\ j \neq i}}^{s} \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=i}} \sum_{\substack{q \in n \\ c(q) \notin \mathcal{F} \\ M(c(q))=j}} \tilde{x}_i \tilde{x}_j \widetilde{w}(n) + \mathbf{d}^t \tilde{\mathbf{x}} + f \\
= \; & 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}(n) x(c(p))^2 - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} \sum_{\substack{q \in n \\ c(q) \notin \mathcal{F} \\ q \neq p}} \widetilde{w}(n) x(c(p)) x(c(q)) + \\
& 4 \sum_{i=1}^{s} \tilde{x}_i \sum_{n \in N} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=i}} \widetilde{w}(n) \Big(\sum_{\substack{q \in n \\ q \neq p}} (\mathrm{ofs}_x(p) - \mathrm{ofs}_x(q)) - \sum_{\substack{q \in n \\ c(q) \in \mathcal{F}}} x(c(q)) \Big) + f \\
= \; & 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} w(n) x(c(p))^2 - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} \sum_{\substack{q \in n \\ c(q) \notin \mathcal{F} \\ q \neq p}} \widetilde{w}(n) x(c(p)) x(c(q)) + \\
& 4 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} x(c(p)) \widetilde{w}(n) \Big((\mathrm{ofs}_x(p) - \mathrm{ofs}_x(q)) \Big) - 4 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} \sum_{\substack{q \in n \\ c(q) \in \mathcal{F}}} x(c(p)) \widetilde{w}(n) x(c(q)) + \\
& \sum_{n \in \mathcal{N}} \sum_{p \in n} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}(n) \Big(\mathrm{ofs}_x(p)^2 + \mathrm{ofs}_x(q)^2 - 2 \mathrm{ofs}_x(p) \mathrm{ofs}_x(q) \Big) + \\
& 4 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} x(c(p)) \widetilde{w}(n) \Big((\mathrm{ofs}_x(p) - \mathrm{ofs}_x(q)) \Big) - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} \sum_{\substack{q \in n \\ c(q) \in \mathcal{F} \\ q \neq p}} \widetilde{w}(n) x(c(q)) x(c(p)) + \\
& 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}(n) x(c(p))^2 \qquad \text{(C.7)} \\
= \; & \sum_{n \in \mathcal{N}} \sum_{p \in n} \sum_{\substack{q \in n \\ q \neq p}} \widetilde{w}(n) \Big(2x(c(p))^2 - 2x(c(p)) x(c(q)) + 4x(c(p)) (\mathrm{ofs}_x(p) - \mathrm{ofs}_x(q)) + \\
& \mathrm{ofs}_x(p)^2 + \mathrm{ofs}_x(q)^2 - 2 \mathrm{ofs}_x(q) \mathrm{ofs}_x(p) \Big) \\
= \; & \sum_{n \in \mathcal{N}} \widetilde{w}(n) \sum_{p \in n} \sum_{\substack{q \in n \\ q \neq p}} (x(c(p)) + \mathrm{ofs}_x(p) - x(c(q)) - \mathrm{ofs}_x(q))^2 \\
= \; & \sum_{n \in \mathcal{N}} CL_{2x}(n) \qquad \text{(C.8)}
\end{aligned}
$$

Note that we use the fact that $(x(c(p)) + \mathrm{ofs}_x(p) - x(c(p)) - \mathrm{ofs}_x(p))^2 = 0$ $\square$

A similar result holds for the star netlength:

Theorem C.2. *Let $M : \mathcal{M} \to \{1, 2, \ldots, s\}$ be a map of the s free modules of a circuit and $N : \mathcal{N} \to \{s+1, \ldots, t\}$ be a map of the $t - s$ nets. Now the x-coordinate of modules and star points can be expressed as a vector $\tilde{\mathbf{x}} \in \mathbb{R}^t$ where the x-coordinate of the module m is $\tilde{x}_{M(m)}$ and the x-coordinate of the net n is $\tilde{x}_{N(n)}$. Further the x-component of the total* star *netlength of a placement can be expressed using matrix-notation as:*

$$\sum_{n \in \mathcal{N}} ST_{2x}(n) \quad = \quad \tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} + \mathbf{d}^t \tilde{\mathbf{x}} + f \tag{C.9}$$

where $\mathbf{C}$ is an $t \times t$ matrix, $d \in \mathbb{R}^t$ and $f \in \mathbb{R}$ are defined as follows:

$$c_{ij} \quad = \quad \begin{cases} \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p)) = i}} w(n) & \text{if} \quad i = j \text{ and } i \leq s \\ \sum_{\substack{n \in \mathcal{N} \\ N(n) = i}} \sum_{p \in n} w(n) & \text{if} \quad i = j \text{ and } i > s \\ \sum_{\substack{n \in \mathcal{N} \\ N(n) = j}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p)) = i}} -w(n) & \text{if} \quad i \leq s \text{ and } j > s \\ \sum_{\substack{n \in \mathcal{N} \\ N(n) = i}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p)) = j}} -w(n) & \text{if} \quad j \leq s \text{ and } i > s \\ 0 & \text{otherwise} \end{cases}$$

$$d_i \quad = \quad \begin{cases} 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p)) = i}} \text{ofs}_x(p) & for \quad i \leq s \quad \text{(i.e. i is module)} \\ -2 \sum_{p \in N^{-1}(i)} \text{ofs}_x(p) - 2 \sum_{\substack{p \in N^{-1}(i) \\ c(p) \in \mathcal{F}}} x(c(p)) & for \quad i > s \quad \text{(i.e. i is net)} \end{cases}$$

$$f \quad = \quad \sum_{n \in \mathcal{N}} \Big(\sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} w(n)(x(c(p))^2 + 2x(c(p))\text{ofs}_x(p)) + \sum_{p \in n} \text{ofs}_x(p)^2 \Big) \tag{C.10}$$

Proof. Again we use the matrix formulation C.9

$$
\begin{aligned}
& \tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} + \mathbf{d}^t \tilde{\mathbf{x}} + f \\
= & \sum_{i=1}^{t} \sum_{j=1}^{t} \tilde{x}_i c_i j \tilde{x}_j + \mathbf{d}^t \tilde{\mathbf{x}} + f \\
= & \sum_{i=1}^{s} \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=i}} w(n) \tilde{x}_i^2 + \sum_{i=s+1}^{t} \sum_{\substack{n \in \mathcal{N} \\ N(n)=i}} \sum_{p \in n} w(n) \tilde{x}_i^2 \\
& -2 \sum_{i=s+1}^{t} \sum_{j=1}^{s} \sum_{\substack{n \in \mathcal{N} \\ N(n)=i}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=j}} w(n) \tilde{x}_i \tilde{x}_j + \mathbf{d}^t \tilde{\mathbf{x}} + f \\
= & \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p))^2 + \sum_{n \in \mathcal{N}} \sum_{p \in n} w(n) st_{2x}(n)^2 - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p)) st_{2x}(n) + \\
& \sum_{i=1}^{t} d_i x_i + f \\
= & \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p))^2 + \sum_{n \in \mathcal{N}} \sum_{p \in n} w(n) st_{2x}(n)^2 - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p)) st_{2x}(n) + \\
& \sum_{i=1}^{s} x_i 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F} \\ M(c(p))=i}} \mathrm{ofs}_x(p) + \sum_{i=s+1}^{s} x_i \Big(-2 \sum_{p \in N^{-1}(i)} \mathrm{ofs}_x(p) - 2 \sum_{\substack{p \in N^{-1}(i) \\ c(p) \in \mathcal{F}}} x(c(p)) \Big) + f \\
= & \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p))^2 + \sum_{n \in \mathcal{N}} \sum_{p \in n} w(n) st_{2x}(n)^2 - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p)) st_{2x}(n) + \\
& 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} x(c(p)) \mathrm{ofs}_x(p) - 2 \sum_{n \in \mathcal{N}} \sum_{p \in n} st_{2x}(n) \mathrm{ofs}_x(p) - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} st_{2x}(n) x(c(p)) + f \\
= & \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p))^2 + \sum_{n \in \mathcal{N}} \sum_{p \in n} w(n) st_{2x}(n)^2 - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} w(n) x(c(p)) st_{2x}(n) + \\
& 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \notin \mathcal{F}}} x(c(p)) \mathrm{ofs}_x(p) - 2 \sum_{n \in \mathcal{N}} \sum_{p \in n} st_{2x}(n) \mathrm{ofs}_x(p) - 2 \sum_{n \in \mathcal{N}} \sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} st_{2x}(n) x(c(p)) + \\
& \sum_{n \in \mathcal{N}} \Big(\sum_{\substack{p \in n \\ c(p) \in \mathcal{F}}} w(n) (x(c(p))^2 + 2 x(c(p)) \mathrm{ofs}_x(p)) + \sum_{p \in n} \mathrm{ofs}_x(p)^2 \Big) \\
= & \sum_{n \in \mathcal{N}} w(n) (x(c(p)) + \mathrm{ofs}_x(p) - st_{2x}(n))^2 \\
= & \sum_{n \in \mathcal{N}} ST_{2x}(n) \qquad \text{(C.11)}
\end{aligned}
$$

□

To solve the unconstrained quadratic problems it is important that the matrices $\mathbf{C}$ from (C.5) and (C.9) are symmetric and positive definite.

Definition C.1. Positive definite matrix. Let A be a square $n \times n$ matrix, then A is positive definite if and only if $x^t Ax > 0$ for any vector $x \neq 0$.

Therefore we prove:

Theorem C.3. *The matrices* $\mathbf{C}$ *from (C.5) and (C.9) are symmetric and positive definite* if *every connected component of the connection graph of the circuit* $\mathcal{C}$ *contains at least one fixed module.*

Proof. Symmetry comes from the definition of $\mathbf{C}$ in theorem C.1 and theorem C.2.

To prove that the matrices are positive definite we define an auxiliary circuit $\mathcal{C}'$ similar to $\mathcal{C}$ but with fixed modules and all pins on fixed modules having their coordinates and offsets set to $(0,0)$. It is easy to see that both $\mathbf{d}$ and f for the auxiliary circuit in (C.5) and (C.9) are $\mathbf{0}$ and 0 respectively. On the other hand $\mathbf{C}$ for $\mathcal{C}$ and $\mathcal{C}'$ must be equal since the location of fixed modules and pins do not affect $\mathbf{C}$.

From theorem C.1 and C.2 we know that the netlength of $\mathcal{C}'$ can be expressed as $\tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}}$. Since the clique and star netlength can only be positive or zero regardless of $\tilde{\mathbf{x}}$ we know that $\tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} \geq 0$ for $\tilde{\mathbf{x}} \in \mathbb{R}^{|\mathcal{M} \setminus \mathcal{F}|}$. This proves that $\mathbf{C}$ is positive semidefinite.

Now assume $\tilde{\mathbf{x}} \neq 0$. I.e. at least one module is not placed at $(0,0)$. Call this module a. Since we assumed all connected components contain at lest one fixed module then a is directly connected or indirectly connected to a fixed module m_f. If a is directly connected to a fixed module m_f then, since all fixed modules are placed at $(0,0)$, the net containing both a and m_f has positive netlength. If a is not directly connected to m_f then there must be a number of intermediate modules $b_1, \ldots, b_n$ with nets (edges) e_i such that $m_f, b_1 \in e_1, \quad b_1, b_2 \in e_2, \quad b_{n-1}, b_n \in e_n$ and $b_n = a$.

We have already proven the statement for $n = 1$. For $n > 1$ there are two cases. Either all b_i, $i < n$ are placed at $(0,0)$ in which case the net (edge) e_n must have positive netlength ($b_n \in e_n$). Otherwise pick the module b_i with smallest i that is not placed at $(0,0)$. Now the netlength of the net (edge) e_i is positive since $b_{i-1} \in e_i$ and $b_i \in e_i$ and b_{i-1} is placed at $(0,0)$.

This proves that $\tilde{\mathbf{x}}^t \mathbf{C} \tilde{\mathbf{x}} > 0$ for $\tilde{\mathbf{x}} \neq 0$. □

Actually the "if" can be replaced with "if and only if" in the theorem (see [24]) but it is of no importance to us.

C.2 Minimizing linear Bounding-Box Netlength

The unconstrained linear bounding box netlength formulation can also be minimized efficiently using network-flow techniques. First we look at the linear program formulation of the bounding box netlength when overlap constraints are omitted.

C.2.1 Linear Program for BB-netlength

A linear program for the BB-netlength can be formulated as:

$$
\begin{array}{lrrcll}
\min & \multicolumn{4}{l}{\sum_{n\in\mathcal{N}} w(n)(\overline{x_n} - \underline{x_n} + \overline{y_n} - \underline{y_n})} & \\
\text{subject to :} & & & & & \\
 & \overline{x_n} & -x(c(p)) & \geq & \text{ofs}_x(p) & n\in\mathcal{N}, p\in n, c(p)\notin\mathcal{F} \\
 & & x(c(p)) \quad -\underline{x_n} & \geq & -\text{ofs}_x(p) & n\in\mathcal{N}, p\in n, c(p)\notin\mathcal{F} \\
 & \overline{x_n} & & \geq & \max_{p\in n}(x(c(p))+\text{ofs}_x(p)) & n\in\mathcal{N}, c(p)\in\mathcal{F} \\
 & & -\underline{x_n} & \geq & -\min_{p\in n}(x(c(p))+\text{ofs}_x(p)) & n\in\mathcal{N}, c(p)\in\mathcal{F} \\
 & \overline{y_n} & -y(c(p)) & \geq & \text{ofs}_y(p) & n\in\mathcal{N}, p\in n, c(p)\notin\mathcal{F} \\
 & & y(c(p)) \quad -\underline{y_n} & \geq & -\text{ofs}_y(p) & n\in\mathcal{N}, p\in n, c(p)\notin\mathcal{F} \\
 & \overline{y_n} & & \geq & \max_{p\in n}(y(c(p))+\text{ofs}_y(p)) & n\in\mathcal{N}, c(p)\in\mathcal{F} \\
 & & -\underline{y_n} & \geq & -\min_{p\in n}(y(c(p))+\text{ofs}_y(p)) & n\in\mathcal{N}, c(p)\in\mathcal{F}
\end{array}
\tag{C.12}
$$

The variables $\overline{x_n}$, $\underline{x_n}$, $\overline{y_n}$ and $\underline{y_n}$ describe bounds for each net $n \in \mathcal{N}$ and the variables $x(m)$ and $y(m)$ describe the lower left coordinate of each module $m \in \mathcal{M}$. All variables are free. Note also the third, fourth, seventh and eight constraint. These represent the pins of the fixed modules.

Since the x-part of objective function and its constraints is completely independent of the y-part the linear program can be split in an x- and y-formulation that can be minimized separately.

Concentrating on the x-part of the linear program we now add constants

$$max_n = \max_{p\in n}(x(c(p)) + \text{ofs}_x(p)) \tag{C.13}$$

$$min_n = \min_{p\in n}(x(c(p)) + \text{ofs}_x(p)) \tag{C.14}$$

and a variable z_0 which is fixed to zero and get:

$$
\begin{array}{lrrcll}
\min & \multicolumn{4}{l}{\sum_{n\in\mathcal{N}} w(n)(\overline{x_n} - \underline{x_n})} & \\
\text{s.t.} & & & & & \\
 & \overline{x_n} & -x(c(p)) & \geq & \text{ofs}_x(p) & n\in\mathcal{N}, p\in n, c(p)\notin\mathcal{F} \\
 & & x(c(p)) \quad -\underline{x_n} & \geq & -\text{ofs}_x(p) & n\in\mathcal{N}, p\in n, c(p)\notin\mathcal{F} \\
 & \overline{x_n} & & \geq & max_n & n\in\mathcal{N} \\
 & & -\underline{x_n} & \geq & -min_n & n\in\mathcal{N} \\
 & & & = & 0 &
\end{array}
\tag{C.15}
$$

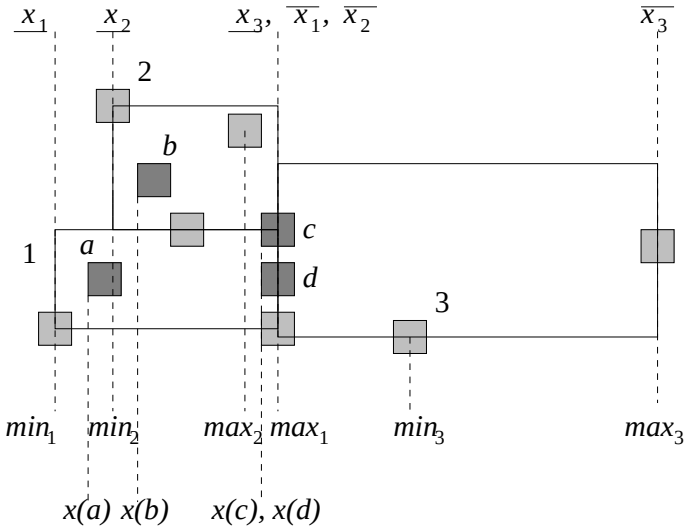


Figure C.1: *Example of the variables and constants of the linear program in (C.15). The figure illustrates a case with three nets* $(1, 2, 3)$ *and four free modules* (a, b, c, d)*. The pale-shaded modules are fixed and the dark shaded modules are free. The net* 1 *connects to modules* a*,* c *and* d*, the net* 2 *connects with modules* b *and* c *and net* 3 *connects with* c *and* d*. All pins are at the center of the modules. The constants* min_1*,* min_2 *and* min_3 *are the x-coordinates of the leftmost pin on the fixed modules of respectively net* 1*,* 2 *and* 3*. Similarly* max_1*,* max_2 *and* max_3 *are the coordinates of the rightmost pin of each net. The variables* $x(a)$*,*$x(b)$*,* $x(c)$ *and* $x(d)$ *are the left coordinates of each module* a, b, c *and* d*. Finally the variables* $\underline{x_1}$*,* $\underline{x_2}$ *and* $\underline{x_3}$ *and* $\overline{x_1}$*,* $\overline{x_2}$ *and* $\overline{x_3}$ *are the x-coordinates of respectively the lower and upper boundaries of each net.*

Figure C.1 shows the meaning of the constants and variables.

The linear program formulation becomes interesting when dualized. Dualizing (C.15) we get:

$$
\begin{array}{ll}
\max & \sum_{n\in\mathcal{N}} \sum_{p\in n, c(p)\notin F} \mathrm{ofs}_x(p)(\overline{f_{pn}} - \underline{f_{pn}}) + \sum_{n\in\mathcal{N}} (max_n \overline{f_n} - min_n \underline{f_n}) \\
\text{s.t.} & \\
& \begin{array}{rcll}
\sum_{p\in n} \overline{f_{pn}} + \overline{f_n} & = & w(n) & n \in \mathcal{N} \\
-\sum_{p\in c^{-1}(m)} \overline{f_{pn(p)}} + \sum_{p\in c^{-1}(m)} \underline{f_{pn(p)}} & = & 0 & m \in \mathcal{M}\setminus\mathcal{F} \\
-\sum_{p\in n} \underline{f_{pn}} - \underline{f_n} & = & -w(n) & n \in \mathcal{N} \\
-\sum_{n\in\mathcal{N}} \overline{f_n} + \sum_{n\in\mathcal{N}} \underline{f_n} + d_0 & = & 0 & n \in \mathcal{N}
\end{array} \\
& \overline{f_{pn}}, \underline{f_{pn}}, \overline{f_n}, \underline{f_n} \geq 0, \quad n \in \mathcal{N}, p \in n^p \\
& d_0 \text{ free}
\end{array}
\tag{C.16}
$$

Here two dual variables $\overline{f_{pn}}$ and $\underline{f_{pn}}$ have been introduced for each pin p not on a fixed module. These correspond to the first two constraints of (C.15). For each net there are dual variables $\overline{f_n}$ and $\underline{f_n}$ corresponding to the third and fourth constraint of (C.15). Finally the variable d_0 correspond to the last constraint and is free.

C.2.2 Network Flow Interpretation

The linear program of (C.16) corresponds to a uncapacitated minimum-cost network flow problem with negative costs. To see this do the following:

- Add two nodes n_s and n_t for each net $n \in \mathcal{N}$.
- Add a node m for each module $m \in \mathcal{M} \setminus \mathcal{F}$
- Add two edges for each pin $p \in n$. One from n_s to $c(p)$ and one from $c(p)$ to n_t.
- Add an extra node d_0 and edges from d_0 to n_t and from n_s to d_0 for each net n.
- Set supply of each node n_s to $w(n)$ and n_t to $-w(n)$ (demand).
- For each edge corresponding to a pin p connecting net n with module m set costs for $C(n_s, c(p)) = -\mathrm{ofs}_x(p)$ and $C(c(p), n_t) = \mathrm{ofs}_x(p)$.
- Set costs for edges (n_t, d_0) to $C(n_t, d_0) = min_n$,
- and costs for edges (d_0, n_s) to $C(d_0, n_s) = -max_n$.

Now solving (C.16) corresponds to solving the above minimum-cost network flow problem. Note that the dual variables $\overline{f_{pn}}$, $\underline{f_{pn}}$, $\overline{f_n}$ and $\underline{f_n}$ corresponds to flow on each edge. Although the dual variable d_0 corresponding to free demand/supply at the node d_0 is free it is easy to see from the flow interpretation that any legal solution will require it to be zero since supply and demand must be equal. Figure C.2 shows how the problem of figure C.1 can be transformed to a network problem.

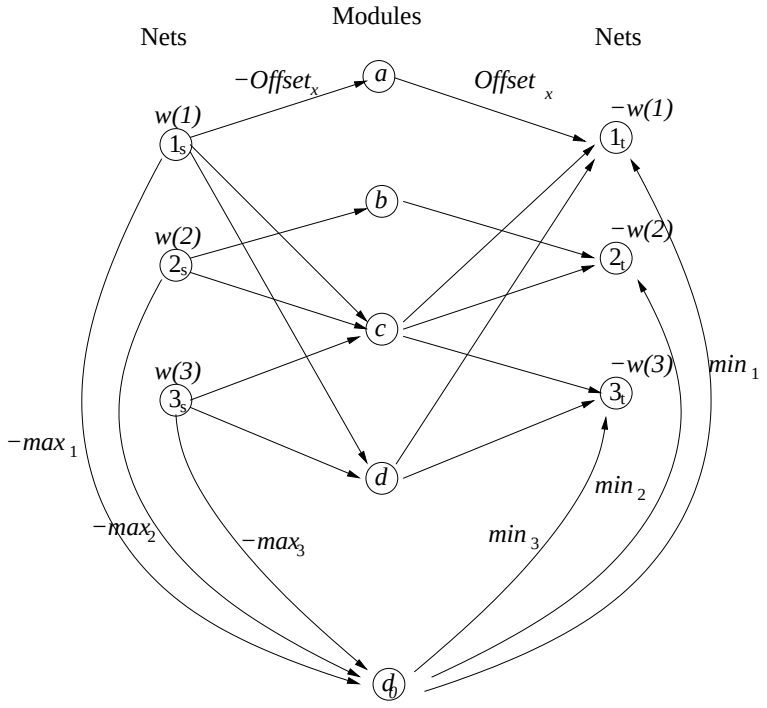


Figure C.2: *The network of figure C.1. There are two nodes for each net and one node for each module. The net nodes 1_s, 2_s and 3_s have supply $w(1)$, $w(2)$ and $w(3)$ respectively. Similarly the nodes 1_t, 2_t and 3_t have demand $w(1)$, $w(2)$ and $w(3)$. The edge cost between a net supply node and module is the offset of the pin connecting the net with module. Similarly the edge cost between a module and net demand node is the negative offset of pin. These have been left out to for clarity. Between each net supply and demand node are edges to a node d_0 The costs on each of these edges corresponds to leftmost and rightmost coordinates of pins connected to fixed modules of the respective nets. section C.2.2*

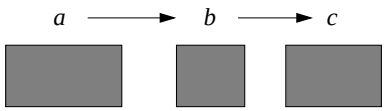


Figure C.3: *If the ordering of the modules is fixed so that a must be left of b and b left of c and the modules are not allowed to overlap then minimizing the linear bounding box netlength can also be solved with network flow methods.*

C.2.3 No-overlap constraints and Fixed Module Sequence

Assume now that the sequence of modules from left to right is fixed and that the modules are not allowed to overlap. For a module b with left adjacent module a and right adjacent module c (see figure C.3) this adds constraints

$$\begin{aligned} x_b - x_a &\geq w_a \\ x_c - x_b &\geq w_b \end{aligned}$$

to the linear program (C.15). Indeed these can also be dualized and will contribute to two more uncapacited edges with costs w_a and w_b in the network flow problem. This shows that this problem is also solvable with network flow methods.

D Semi-Convex Functions

In this appendix we describe a number of properties for a special kind of piece-wise linear functions. We have found no literature on this subject and have therefore theorized on our on. We have called this special kind of linear functions semi-convex functions. And we begin with the following definition.

Definition D.1. Slope of linear function The slope of a linear function $f(x)$ defined on an interval I is

$$f^s = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \tag{D.1}$$

for $x_1, x_0 \in I$ and $x_1 \neq x_0$.

Since $f(x)$ is linear the slope is independent of x_1 and x_0.

Definition D.2. Slope of a continuous piece-wise linear function For a piece-wise continuous linear function $f(x) : \mathbb{R} \to \mathbb{R}$ defined as:

$$f(x) = \begin{cases} f_1(x) & \text{for} \quad x \in]\tilde{x}_0, \tilde{x}_1] \\ f_2(x) & \text{for} \quad x \in]\tilde{x}_1, \tilde{x}_2] \\ \cdots & \cdots \quad \cdots \\ f_k(x) & \text{for} \quad x \in]\tilde{x}_{k-1}, \tilde{x}_k[\end{cases}, \tag{D.2}$$

with $\tilde{x}_0 = \infty$ and $\tilde{x}_k = \infty$ the slope $f^s(x)$ at x is defined as f_j^s for $x \in]\tilde{x}_{j-1}, \tilde{x}_j]$.

Lemma D.1. *For a continuous function $h(x) : \mathbb{R} \to \mathbb{R}$ which is the sum of two continuous piece-wise linear functions $f(x) : \mathbb{R} \to \mathbb{R}$ and $g(x) : \mathbb{R} \to \mathbb{R}$, $h(x) = f(x) + g(x)$, we have $h^s(x) = f^s(x) + g^s(x)$.*

Proof. $h(x)$ is piece-wise linear since it is the sum of two piece-wise linear functions therefore $h(x)$ consists of piece-wise linear functions h_j defined on intervals $]\tilde{x}_{j-1}, \tilde{x}_j]$. If $x \in]\tilde{x}_{j-1}, \tilde{x}_j]$ for some interval j then $h^s(x)$ is defined as the slope of the linear function h_j. Since $h(x)$ is a sum of piece-wise linear functions, we must have $h_j(x) = f_i(x) + g_l(x)$ for appropriate i and l. As $h_j(x)$ is linear we may choose $x_0, x_1 \in]\tilde{x}_{j-1}, \tilde{x}_j]$, $x_0 \neq x_1$ and we have that:

$$\begin{aligned} h_j^s &= \frac{h_j(x_1) - h_j(x_0)}{x_1 - x_0} \\ &= \frac{f_i(x_1) + g_l(x_1) - f_i(x_0) - g_l(x_0)}{x_1 - x_0} \\ &= f_i^s + g_l^s \\ &= f^s(x) + g^s(x). \end{aligned} \tag{D.3}$$

Since $f^s(x)$ and $g^s(x)$ are independent of j we may write $h^s(x) = f^s(x) + g^s(x)$. □

We now introduce a property for continuous linear functions which we call semi-convex.

Definition D.3. Semi-convex A function $f(x)$ is said to be *semi-convex* if:

1. $f(x)$ is piece-wise linear,
2. $f(x)$ is continuous,
3. $f^s(x) \geq f^s(x_0)$ for all $x \geq x_0$, and
4. $f(x) \to \infty$ for $x \to -\infty$ and $x \to \infty$.

Lemma D.2. *The sum of two semi-convex functions is also semi-convex.*

Proof. Assume a piece-wise linear function $h(x) : \mathbb{R} \to \mathbb{R}$ is the sum of semi-convex functions $f(x) : \mathbb{R} \to \mathbb{R}$ and $g(x) : \mathbb{R} \to \mathbb{R}$. We must prove the four conditions of definition D.3.

1, 2. The sum of two continuous and piece-wise linear functions is continuous and piece-wise linear.

3. Since $h^s(x) = g^s(x) + f^s(x)$ we have

$$h^s(x) = g^s(x) + f^s(x) \geq g^s(x_0) + f^s(x_0) = h^s(x_0), \tag{D.4}$$

for $x \geq x_0$.

4. Since $f(x) \to \infty$ and $g(x) \to \infty$ for $x \to \infty$ and $x \to -\infty$ we have $h(x) = f(x) + g(x) \to \infty$ for $x \to \infty$ and $x \to -\infty$. □

Semi-convex functions have the following useful property:

Lemma D.3. *Assume $f(x)$ is semi-convex then:*

$$\frac{f(x_0 + h) - f(x_0)}{x_0 + h - x_0} \quad \geq \quad f^s(x_0) \tag{D.5}$$

for $x_0 \in \mathbb{R}$ and $h > 0$, and

$$\frac{f(x_0) - f(x_0 - h)}{x_0 - (x_0 - h)} \leq f^s(x_0), \tag{D.6}$$

for $x_0 \in \mathbb{R}$ and $h > 0$.

Proof. Assume that $f(x)$ is broken into its linear components $f_i(x)$ defined on intervals $]\tilde{x}_{i-1}, \tilde{x}_i]$. Now let $x_0 \in]\tilde{x}_{j-1}, \tilde{x}_j]$ and $x_0 + h \in]\tilde{x}_{k-1}, \tilde{x}_k]$. Since $f(x)$ is continuous and

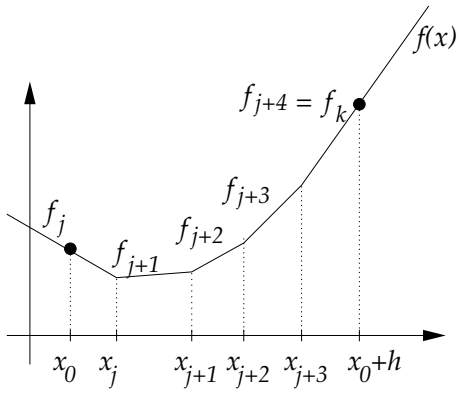


Figure D.1: *The expansion of $f(x)$ from x_0 in lemma D.3.*

piece-wise linear we have $f(x) = f^s_{j-1} \cdot (x - x_{j-1}) + f(x_{j-1})$ for $x \in]x_{j-1}, x_j]$, so we can expand $f(x_0)$ from x_0 (see figure D.1) and get:

$$\begin{aligned} f(x_0 + h) &= f(x_0) + \sum_{i=j+1}^{k-1} f^s_i \cdot (\tilde{x}_i - \tilde{x}_{i-1}) + f^s_j \cdot (\tilde{x}_j - \tilde{x}_0) + f^s_k \cdot (\tilde{x}_0 + h - \tilde{x}_{k-1}) \\ &\geq f(x_0) + f^s_j \left(\sum_{i=j+1}^{k-1} \cdot (\tilde{x}_i - \tilde{x}_{i-1}) + (\tilde{x}_j - x_0) + (x_0 + h - \tilde{x}_{k-1}) \right) \\ &= f(x_0) + f^s_j (x_0 + h - x_0), \end{aligned} \tag{D.7}$$

since $f^s_j \leq f^s_i$ for $i > j$. Using this relation we get

$$\frac{f(x_0 + h) - f(x_0)}{x_0 + h - x_0} \geq f^s_j = f^s(x_0). \tag{D.8}$$

which proves the first statement. The second statement is proven similarly. □

The previous lemma allows us to prove the following nice property for semi-convex functions.

Lemma D.4. *Assume $f(x) : \mathbb{R} \to \mathbb{R}$ is semi-convex. Then for $x_0 \in \mathbb{R}$ and $f(x_0) = c$ there exists x_r and x_l such that $f(x) > c$ for $x \in \mathbb{R} \setminus [x_l, x_r]$, and $f(x) \leq c$ for $x \in [x_l, x_r]$.*

Proof. Since $f(x) \to \infty$ for $x \to -\infty$ and $x \to \infty$ we can choose

$$\begin{aligned} x_l &= \max\{x_0 | f(x) > c \text{ for all } x < x_0\} \\ x_r &= \min\{x_0 | f(x) > c \text{ for all } x > x_0\} \end{aligned} \tag{D.9}$$

By this choice we have $f(x) > c$ for $x \in \mathbb{R} \setminus [x_l, x_r]$, so we must prove $f(x) \leq c$ for $x \in [x_r, x_l]$. Assume that there exists x_0 such that $f(x_0) > c$ and $x_0 \in]x_l, x_r[$. Now

choose $h_1 > 0$ such that $f(x_0 - h_1) \leq c$ and $h_2 > 0$ such that $f(x_0 + h_2) \leq c$. h_1 and h_2 exist since $f(x_l) \leq c$ and $x_l < x_0$, and $f(x_r) \leq c$ and $x_r > x_0$.

This gives us:

$$\begin{aligned} \frac{f(x_0 + h_2) - f(x_0)}{x_0 + h_2 - x_0} &< 0, \\ \frac{f(x_0) - f(x_0 - h_1)}{x_0 - (x_0 - h_1)} &> 0, \end{aligned}$$

since $f(x_0) > c$, $f(x_0 - h_1) \leq c$ and $f(x_0 + h_2) \leq c$. Which means

$$\frac{f(x_0 + h_2) - f(x_0)}{x_0 + h_2 - x_0} < \frac{f(x_0) - f(x_0 - h_1)}{x_0 - (x_0 - h_1)} \tag{D.10}$$

This contradicts lemma D.3 which says:

$$\frac{f(x_0) - f(x_0 - h_1)}{x_0 - (x_0 - h_1)} \geq \frac{f(x_0 + h_2) - f(x_0)}{x_0 + h_2 - x_0}. \tag{D.11}$$

Therefore no such x_0 can exist if $f(x)$ is semi-convex. □

E User manual

This is the user manual for the G/Flegal global- and final-placement program. The complete source-code is available at

```
www.diku.dk:/hjemmesider/studerende/jegeblad/vlsi.zip
```

E.1 Compiling G/FLegal

A `tmake` project-file is provided with G/FLegal which can be used to create a regular `makefile`. The `tmake`-file, gflegal.pro, requires tmake which is available at `www.trolltech.com`. Within the `tmake`-file are several parameters that can be adjusted. These should be self-explanatory. Gcc v. 3.0 or higher is required for compilation. Also required is QT v. 2.3 or higher. QT for Linux and Windows is available at `www.trolltech.com`. Finally for relaxation based local search either CPlex or GLPK is required. GLPK is available at `www.gnu.org/software/glpk/glpk.html`.

E.2 Invoking G/FLegal

The program can run in both interactive graphical mode and in non-interactive command-line mode. We will only describe the command-line mode here. The standard command-line format is:

```
GFLegal [inputfile] [option-1] ... [option-n]
```

where `[inputfile]` is the filename of the VLSI G/FLegal should optimize. Acceptable files may be YAL-format, the XML-format of [25] or our own properiatary and binary RAW-format. Options often accept a parameter value which should follow the option (no space). Example:

```
GFLegal primary2.xml -Oregionnet -Aauto
```

E.2.1 Command-line Options to Determine Operation-Mode

G/Flegal can run in two basic modes: non-interactive and interactive. G/Flegal can also be invoked to create new files and save best solutions. The possible options are shown in

Option	Meaning
`-help`	Displays a list of command-line options and their meaning
`-n`	Non-interactive mode (default mode is interactive)
`-r`	Convert instance to RAW-format once loaded and exit
`-s`	Output statistics of circuit
`-newcopy`	Create a 32×32 tiled copy of the instance as described in section 5.3 and saves it. Parameter is output-filename.
`-makegeneral`	Converts a standard-cell instance to a general-cell instance as described in section 5.3
`-right`	Moves the right boundary and all pins on it $p\%$ to the right. p is parameter
`-play`	Play mode. Allows the user to do more things during interactive mode.
`-savebest`	Saves the best solution encountered during optimization. Parameter is a tag to appended the instance name and the output-format is `[filname]_[parameter]_best.raw`.
`-savetenth`	Saves solutions at every tenth iteration to an individual file. output-name is `[filename]_[parameter]_[iteration].raw`
`-savelast`	Saves the last solution. `[filename]_[parameter]_last.raw`
`-savebestsp`	Saves the best solution encountered during legalization. Output-name is `[filename]_[parameter]_bestsp.raw`
`-I`	Stops the optimization after n iterations. n is parameter
`-seconds`	Stops the optimization after n seconds. n is parameter

E.2.2 Command-line Parameters for Optimization

G/FLegal contains numerous parameters for optimization. These are

Option	Meaning
`-O`	Optimization method. Determines which method G/FLegal optimizes by. Documented methods are `generic` (Force-based method by Eisenmann and Johannes [19]), `regionnet` (the optimization method of section 7). `simpleswap` (the clean-up step of global placement) and `relaxedls` (relaxation based local search of section 8). G/Flegal also include a number of undocumented optimization methods which were used during preliminary testing and may be in a unstable state with the current implementation.
`-S`	Sequence-pair origin placement strategy. Parameters are `lowerleft` (place modules in lower-left corner of placement area), `centersim` (place modules in center of placement area. Split modules in four regions at the center of the placement area). `centermed` (place modules in center of placement area. Split modules in four approximately equal size groups based before placing).
`-E`	Sequence-pair layout technique. `envelope` uses the standard envelope of Pisinger [72]. `envelopeext` uses our extended envelope technique. `lcs` uses the longest common sub-sequence method to place modules.
`-Q`	Chooses quadratic function. Parameters are `clique`, `star`
`-linearization`	Enables or disables linearization scheme
`-densitygrid`	Determines the grid resolution of Eisenmann and Johannes' force based approach. Parameter is resolution level n. Grid will have resolution 2^n.
`-seed`	Random seed to use
`-areatest`	Does area compaction tests.
`-areaopt`	Does one run of area compaction.
`-A`	Set value of α in sequence-pair algorithm or select `auto` to search through 20 different values of α for each legalization.
`-gamma`	Set value of γ in augmented objective function.
`-delta`	Set value of δ for global placement.
`-omega`	Set value of ω for global placement.
`-sak`	Set simulated annealing coolage value.
`-sastart`	Set simulated annealing start time.
`-legmoves`	Set the fraction of modules to be moved before legalization in each step.
`-resetinterval`	Number of iterations between reset of simulated annealing.
`-grow`	Grow modules during regionnets optimization.

E.2.3 Command-line Options for Debug-Output

G/Flegal also contains debug output. This can be turned on but it requires that the program has been compiled with `-DDEBUG`. The debug options are

Option	Meaning
`-nomessages`	During non-interactive mode some extra messages are output. This option turns off these messages.
`-debug`	Outputs various debug information. Debug information has been split into several groups. `all` Gives all debug output. `algorithm` gives output from algorithms. `optimization` gives output from the optimization process. `timer` gives varous run-times. `gui` gives output from the graphical user interface. `none` gives no debug output.