Algorithms for Improving the Design and Production of Oligonucleotide Microarrays

Sérgio Anibal de Carvalho Junior

January 2007

PhD thesis submitted to the
Faculty of Technology of Bielefeld University, Germany,
for the degree of Dr. rer. nat.

Supervisor:
Dr. Sven Rahmann

1. Referee: Prof. Dr. Jens Stoye
2. Referee: Dr. Sven Rahmann

Foreword

Microarrays are a ubiquitous tool in molecular biology with a wide range of applications on a whole-genome scale including high-throughput gene expression analysis, genotyping, and resequencing.

Several different microarray platforms exist, but this thesis focuses on high-density oligonucleotide arrays. The advantage of higher densities arrays is that they allow, for instance, the simultaneous measurement of the expression of several thousand genes at once.

High-density microarrays are usually produced by light-directed combinatorial chemistry that builds the probe sequences base-by-base. Because of the natural properties of light, the quality of a microarray can be improved by carefully designing the physical arrangement, or *layout*, of its probes.

In this thesis, we review models for evaluating the layout of oligonucleotide microarrays and survey algorithmic approaches that can be used in their design.

DNA chips allow to quickly obtain and compare gene expression profiles of different cell or tissue samples. As one of many applications, one hopes to better understand various types and subtypes of cancer and to improve cancer therapy by characterizing the differences between the expression profiles of healthy cells and tumor cells.

The first chapter of this thesis offers an overview of existing microarray technologies and contrasts them with other techniques for gene expression analysis. High-density oligonucleotide arrays are described in detail.

Gene expression analysis with DNA chips is a high-throughput technique that produces massive amounts of data; however, this technique is not error-free. Each step of an experiment must be performed carefully. In particular, the DNA chip must be carefully designed and manufactured. This thesis proposes and describes solutions for some of the algorithmic problems that arise in this phase. These problems are formulated in detail in the last section of Chapter x.

My research started with the general question of how to find oligonucleotide probes for highly homologous transcripts when it cannot be guaranteed that a sufficiently large set of clearly transcript-specific (or *unique*) probes can be found. An interesting future large-scale application could be the individual expression measurement of all splice variations of all genes in the human genome, for example. The basic idea was to find several *non-unique* probes such that different probes hybridize to different combinations of targets, with the hope that the measured signal can then be decoded into the individual expression levels. Two question then follow immediately: How does the decoding work, and how can the probes be designed in a way that makes the decoding as simple as possible?

In early 2001, custom probe design was in its infancy. The design of the large commercial chips, such as the Affymetrix GeneChip®, was a well guarded secret of the respective companies. Several other research groups were also beginning to work on probe selection, and their results, made the problem more popular. I soon realized that, before actually working on non-unique probes, more fundamental questions should be addressed. A high-performance large-scale probe selection system for unique probes would be very useful but did not exist. Such a system could then be used as a basis for non-unique probe design. These considerations are reflected in this thesis.

Once a set of probes is chosen for a chip design, the chip must be produced. With several technologies, such as the Affymetrix GeneChip® arrays and febit's geniom® one system, the probes are synthesized in situ on the chip with a combination of photolithography and combinatorial chemistry. We consider the problem of optimizing the nucleotide deposition sequence to lower production costs and to decrease the overall error rate during synthesis.

A concluding discussion about the results of this thesis and an outlook into the future can be found in Chapter 10.

Publications. Parts of this thesis have been published in advance.

This thesis also contains previously unpublished material, namely

- new placement algorithm that simultaneously places and re-embeds the probes

Most of the material was published as a book chapter in de Carvalho Jr. and Rahmann (2007).

The stuff on the SCS appeared in de Carvalho Jr. and Rahmann (2005).

Acknowledgments. This work was carried out while I was at the... It has been a very enjoyable experience, thanks to all present and former colleagues and students in the Bioinformatics program.

I would especially like to thank Sven Rahmann for suggesting the topic, providing initial ideas, and for the opportunity to write this thesis under his guidance.

The support of the whole computer service group at the CeBiTec was crucial to the success of this work, and I cannot thank them enough for their help.

I enjoyed fruitful discussions with many people while this thesis was underway; in particular, I am grateful to ... for their valuable hints and comments.

Ferdinando Cicalese, José Augusto Amgarten Quitzau, Klaus-Bernd Schürmann, ... read early drafts of several chapters of this thesis and helped to improve it in many ways through their comments.

I would like to thank Dr. Peter Hahn and Chris MacPhee for working on several QAP instances derived from artificial microarrays from Chapter 4 and for helpful discussions.

Sérgio A. de Carvalho Jr. Bielefeld, January 2007

Danke schön!

Danke schön!

Abbreviations and Notation

Abbreviations in alphabetical order

A, `A`	adenine
Å	Angstrom; 1Å = 10^{-10} m = 1/10 nm
aRNA	anti-sense RNA, amplified RNA; complementary to mRNA
bp	base pair(s)
C, `C`	cytosine
°C	degrees centigrade (Celsius)
cDNA	complementary DNA; DNA synthesized from mRNA by RT
CDS	coding sequence
cRNA	complementary RNA; same as aRNA
CSMS, csms	cumulative statistics of matching statistics
Cy3	Cyanine 3-dNTP; in fluorescently labeled DNA
Cy5	Cyanine 5-dNTP; in fluorescently labeled DNA
DMD	digital micromirror device
DNA	deoxyribonucleic acid
dN	any deoxynucleotide; any of dA, dC, dG, dT
dNTP	any deoxynucleotide-triphosphate; any of dATP, dCTP, dGTP, dTTP
EST	expressed sequence tag
G, `G`	guanine
HPLC	high-pressure liquid chromatography
IVT	in-vitro transcription
K	Kelvin; physical unit of temperature
LCF	longest common factor
LCFS	longest common factor statistics
M	physical unit of molar concentration: 1 M = 1 mol/l
MCMC	Markov chain Monte Carlo
MES	2-(N-morpholino)ethanesulfonic acid
mol	physical unit of amount of substance 1 mol = as many entities as atoms in 0.012 kg of carbon 12
mRNA	messenger RNA
MS, ms	matching statistics
N, `N`	any of the bases A, C, G, T/U
[Na^+]	salt (sodium ion) concentration
NaCl	salt (sodium chloride)
NN	nearest neighbor
nt	nucleotide(s)

oligo-(dT)	primer of 12–20 dTs binding to the poly(A) tail of eukaryotic mRNA
P	phosphor
^{33}P	a radioactive phosphor isotope
PAGE	polyacrylamide gel electrophoresis
PCR	polymerase chain reaction
PEM	percent of expected mutations (unit of evolutionary time)
RNA	ribonucleic acid
RT	reverse transcription
RTase	reverse transcriptase; an enzyme
RT-PCR	reverse transcription-polymerase chain reaction
SAGE	serial analysis of gene expression
SAPE	streptavidin phycoerythrin
SBH	sequencing by hybridization
SCS	shortest common supersequence
SCSP	shortest common supersequence problem
SNP	single nucleotide polymorphism
SSC	saline sodium citrate buffer
SSPE	saline sodium phosphate buffer
SVD	singular value decomposition
T, `T`	thymine
T7	a bacteriophage that infects *E. coli*; these viral parasites are numbered T1 through T7
Taq	*Thermus aquaticus*; a heat-resistant bacterium
TIFF	tagged image file format
transfrag	transcript fragment
tRNA	transfer RNA
U, `U`	uracil
UTR	untranslated region; part of mRNA transcripts

Notation by topic

Notation introduced in early chapters is also used in subsequent chapters.

Chapter 1

p	probe sequence; a string
$\|p\|$	length of p
t	transcript or target sequence
T	transcriptome $T = (t_1, \ldots, t_n)$
m	number of probes
i	probe index
n	number of transcripts
j	transcript index
θ	hybridization conditions; parameters
$a(p, t; \theta)$	affinity coefficient between p and t, given θ

$p \lhd t$	p is a factor (substring) of t
ε	lower affinity limit for specific hybridization

Chapter 2

$x = (x_j)$	n-vector of transcript expression levels
$y = (y_i)$	m-vector of measured probe signals
$A = (A_{ij})$	$m \times n$-matrix of probe-transcript affinity coefficients; $y = A \cdot x$
$t(i)$	index of the unique target that probe i binds to
a, a_i	particular affinity coefficients
ρ_{ij}	position dependent factor of A_{ij}
β_{ij}	hybridization stability dependent factor of A_{ij}
γ_{ij}	sequence composition dependent factor of A_{ij}
σ_{ij}	self-complementarity dependent factor of A_{ij}
ε_i	stochastic additive noise of probe signal intensity
c_i, c	systematic additive offset of signal intensity
α	scale of signal intensity
η_{ij}	stochastic multiplicative noise of signal intensity
d	differential operator
Δ	denotes a difference
U; q; w	internal energy; heat; work
p	pressure
V	volume
T	temperature
H; H°_m; $\Delta_\mathrm{r}H^\circ$	enthalpy; standard molar enthalpy; standard reaction enthalpy
S; S°_m; $\Delta_\mathrm{r}S^\circ$	entropy; standard molar entropy; standard reaction entropy
G; G°_m	Gibbs energy; standard molar Gibbs energy
$\Delta_\mathrm{r}G^\circ$; $\Delta_\mathrm{r}G$	standard Gibbs energy of reaction; reaction Gibbs energy
ξ	extent of reaction
R	gas constant; $R = 8.3145$ J K^{-1} mol^{-1}
K	equilibrium constant
$[\mathrm{S}_2]_\mathrm{eq}$	equilibrium concentration of reactant S_2 (single stranded probes)
T_M	melting temperature

Chapter 3

$\mathrm{lcf}(p, t)$	length of the longest common factor of p and t
$\mathrm{lcf}^1(p, t)$	same, allowing one mismatch
$\mathrm{lcf}^*(p, t)$	combined measure derived from lcf and lcf^1
$\mathrm{LCF}(p \mid T)$	LCF vector of probe p against transcriptome T
$\mathrm{LCFS}(p \mid T; \Delta)$	LCF statistics of p against T of width Δ
δ	index of LCF statistics, $0 \le \delta < \Delta$ difference between full probe length and LCF length
$T(i)$	index set of intended targets for probe i
T	transcriptome; see Chapter 1
$U'(p_i \mid T)$	unspecificity of probe i in T; formal definition
τ	temperature; to avoid confusion with transcriptome T

$\beta(\delta)$	average hybridization probability as a function of δ
ζ	offset constant; $\zeta = \ln(\beta(0)/(1-\beta(0)))$
$b > 0$	average Gibbs energy per base pair in units of $(R\tau)$
$u(\delta)$	approximation to $\beta(\delta)$
$U(p_i \mid T)$	unspecificity of probe i in T; practical definition

Chapter 4

$T = (T_c)$	transcriptome of transcript collections T_c $(c = 1..C)$
C	number of collections
$T_c = \{t_{c,j}\}$	transcript collection c with transcripts $t_{c,j}$ $(j = 1..N_c)$
N_c	number of transcripts in collection c
s	target sequence
Σ	alphabet; here $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$
Σ^*; Σ^+	all strings (non-empty strings) over alphabet Σ
`$`	string end marker and separator
`X`	unspecified or wildcard character
`pos`	suffix array
$\mathrm{lcp}(s,t)$	length of the longest common prefix of s and t
`lcp`	longest common prefix array
q	bucket prefix length
`bck`	bucket array
$\gamma = \langle Q \rangle$	numerical radix-q representation of a q-gram Q
`cl`	collection number array
$\mathrm{ms}^{s\|t} = (\mathrm{ms}_i^{s\|t})$	matching statistics of s against t
$\mathrm{ms}^{s\|t;f}$	matching statistics allowing f mismatches of s against t
$R^0_{\min}$	minimum value of relevant matching statistics
$R^1_{\min}$	minimum value of relevant ms. with one mismatch
$R_{\max}$	maximum value of relevant matching statistics
$\texttt{MS}[i][c]$	matching statistics array; $\texttt{MS}[i][c] = \mathrm{ms}_i^{s\|T_c}$
(i, J)	jump at position i to level J
n, m	string lengths; $\|s\| = m$, $\|t\| = n$
$\mathbb{P}$	generic notation for a probability measure
$\mathbb{E}$	generic notation for an expectation
$\pi = (\pi_c)_{c\in\Sigma}$	character distribution for a random sequence model
p	probability that two random characters match
q	$q := 1 - p$
K_L	random number of occurrences of a random L-gram in a string
ρ_L	probability that a jump to level L occurs at a fixed position
E_L	expected number of jumps to level L in matching statistics
E_L^+	expected number of jumps to level $\geq L$ in matching statistics
L°	center of the distribution of the LCF of two random strings

Chapter 5

d	probe distance from the transcript's $3'$-end

$p_i = (d_i, \ell_i)$	probe i defined by end distance d_i and length ℓ_i
$f(d)$; f_i	position preference factor for distance d; $f_i := f(d_i)$
h_i	hairpin formation probability for probe i
U_i	unspecificity for probe i
B_i	badness value for probe i
$\mathcal{B}(\delta)$	additional badness for probes clustering within distance δ
B'_i; $\overline{B}'$	modified badness value for probe i; threshold $\overline{B}'$
S	selected probe set

Chapter 6

$\mathbb{1}_{\{\cdot\}}$	generic notation for an indicator function
$S = (s_c)$	target sequences for all collections $c = 1, \dots, C$
$H = (H_{ij})$	binary $m \times n$ hybridization matrix of probes vs. targets
D	design, i.e. a selection of probes; $D \subset \{1, \dots, m\}$
A^D; H^D	affinity and hybridization matrix restricted to rows from D
$\mathcal{M}$	minimum target coverage
$\mathcal{A}$	average target coverage
A^T	transpose of A
Σ	diagonal matrix with singular values (σ_j) of A
σ_j	j-th largest singular value
$\mathrm{diag}(\cdot)$	diagonal matrix with specified entries
A^-	pseudo-inverse of A
$\|\cdot\|_2$	Euclidean norm of a vector; spectral norm of a matrix
$\|\cdot\|_\infty$	maximum norm of a vector
$\mathrm{cond}(A)$	condition number of A
$\mathrm{cond}_\mathrm{s}(A)$	Skeel condition number of A
μ	maximal number of probes that may be selected
S,T	sets of target indices
$S \Delta T$	symmetric set difference of S and T
$P(S)$	set of probes that hybridize to any target in S
$T(i)$	set of targets that hybridize to probe i
γ	desired minimum target coverage
σ	desired minimum target set separation
$\delta \in \{0,1\}^m$	binary vector representation of design D
h_j	j-th column of binary hybridization matrix H
$z^{S,T} = (z_i^{S,T})$	indicates whether probe i separates sets S and T
δ^+, h_j^+, z^+	vectors extended for virtual unique probes
B	maximal size of target sets to be separated
f_0; f_1	false negative (positive) error rate
$\mathbb{P}(x)$	prior probability of binary expression vector $x \in \{0,1\}^n$
$\mathbb{P}(y \mid x)$	likelihood of probe signals $y \in \{0,1\}^m$, given x
$\pi(x) = \mathbb{P}(x \mid y)$	posterior probability of x for fixed y
$q(z \mid x)$	proposal probability for new solution z at current solution x
$\mathcal{N}(x)$	neighborhood of x

$\alpha(x,z)$	acceptance probability for z at x
Q	rate matrix of an evolutionary Markov process
t	evolutionary time
P^t	Markov transition kernel for time t
$\mathrm{expm}(Q)$	matrix exponential; $\mathrm{expm}(Q) := \sum_{n\geq 0} Q^n/n!$

Chapter 7

$\mathrm{csms}^{s\|t}$	cumulative statistics of matching statistics of s against t $\mathrm{csms}^{s\|t}_{i,\mu}$ refers to position i in s and matches of length $\geq \mu$
$\mathtt{CSMS}[i][\mu]$	CSMS array; $\mathtt{CSMS}[i][\mu] = \mathrm{csms}^{s\|t}_{i,\mu}$ if $\mu \in [R^0_{\min}, R_{\max}]$
$\sigma^i_{k,u}$	unexpected CSMS at offset k for a probe starting at i
$L_{i,\delta}$	whole genome surrogate for $\mathrm{LCFS}(p_i)_\delta$
L	length of hypothetical transfrags
D	distance between start points of hypothetical transfrags
K	transfrags known to be present
P_1	probes expected to show a signal because of K
M	transfrags of unknown status; $M = \{1..n\} \setminus K$
N	transfrags most likely not present
J	transfrags whose status is inferred by sampling; $J = \{1..n\} \setminus (K \cup N)$
χ	fraction of transfrags hybridizing to additional probes

Chapter 8

Σ	DNA alphabet
π	permutation of the nucleotides
$\mathcal{S}$	sequence set
e	binary embedding vector of a string into a supersequence
E	embedding matrix with rows e_i, $i = 1..\|\mathcal{S}\|$
$\mathcal{U}$	upper bound on the supersequence length
$\mathcal{L}$	lower bound on the supersequence length
L_i	length of the i-th sequence in $\mathcal{S}$
$N_i(x)$	number of occurrences of $x \in \Sigma$ in the i-th sequence
$N(x)$	$\max_i N_i(x)$
C_i	completion step of the i-th sequence
$W_{i,k}$	number of unproductive steps between $(k-1)$-st and k-th productive step for sequence i
Φ	cumulative distribution function of the standard Normal distribution

Contents

Chapter 1

Introduction

1.1 DNA and Genes

Choose another title for this sub-section.

1.2 Functional Genomics

Choose another title for this sub-section.

1.3 High-density Oligonucleotide Microarrays

Oligonucleotide microarrays consist of short DNA fragments, called *probes*, affixed or synthesized at specific locations, called *features* or *spots*, of a solid surface. Microarrays are based on the principle of Watson-Crick base pairing. Each probe is a single-stranded DNA molecule of 10 to 70 nucleotides that perfectly matches with a specific part of a *target* molecule. The probes are used to verify whether (or in which quantity) the targets are present in a given biological sample.

This type of microarray was originally designed in the late 1980s as a tool for DNA sequencing, a technology that is known as DNA Sequencing by Hybridization.

TODO: mention uniform arrays.

The first step of a microarray experiment consists of collecting mRNAs or genomic DNA from the cells or tissue under investigation. The mixture to be analyzed is prepared with fluorescent tags and loaded on the array, allowing the targets to hybridize with the probes. Any unbound molecule is washed away, leaving on the array only those molecules that have found a complementary probe. Finally, the array is exposed to a light source that induces fluorescence, and an optical scanner reads the intensity of light emitted at each spot.

Under ideal conditions, each probe will hybridize only to its target. Thus, it is possible to infer whether a given molecule is present in the sample by checking whether there is light coming from the corresponding spot of the array. The expression level of a gene in a cell can also be inferred because each spot contains several million identical probes, and the strength of the fluorescent signal on a spot is expected to be proportional to the concentration of the target in the sample. In practice, each target is queried by several probes (its *probe set*), and complex statistical calculations are performed to infer the concentration from the observed signals.

High-density microarrays, also called microarray chips, can have more than a million spots, and are thus able to query tens of thousands of genes, covering the entire genome of an organism. The pioneering Affymetrix GeneChip® arrays, for instance, have up to 1.3 million spots on a coated quartz substrate measuring a little over 1 cm^2. The spots are as narrow as 5 μm (5 microns, or 0.005 mm), and are arranged in a regularly-spaced rectangular grid.

1.3.1 Microarray Production with Photolithographic Masks

GeneChip arrays are produced by combinatorial chemistry and techniques derived from micro-electronics and integrated circuits fabrication. Probes are typically 25 bases long and are synthesized on the chip, in parallel, in a series of repetitive steps. Each step appends the same kind of nucleotide to probes of selected regions of the chip. The selection of which probes receive the nucleotide is achieved by a process called *photolithography* (Fodor et al., 1991, 1993).

Figure 1.1 illustrates this process: The quartz wafer of a GeneChip array is initially coated with a chemical compound topped with a light-sensitive protecting group that is removed when exposed to ultraviolet light, activating the compound for chemical coupling. A lithographic mask is used to direct light and remove the protecting groups of only those positions that should receive the nucleotide of a particular synthesis step. A solution containing adenine (A), thymine (T), cytosine (C) or guanine (G) is then flushed over the chip surface, but the chemical coupling occurs only in those positions that have been previously deprotected. Each coupled nucleotide also bears another protecting group so that the process can be repeated until all probes have been fully synthesized.

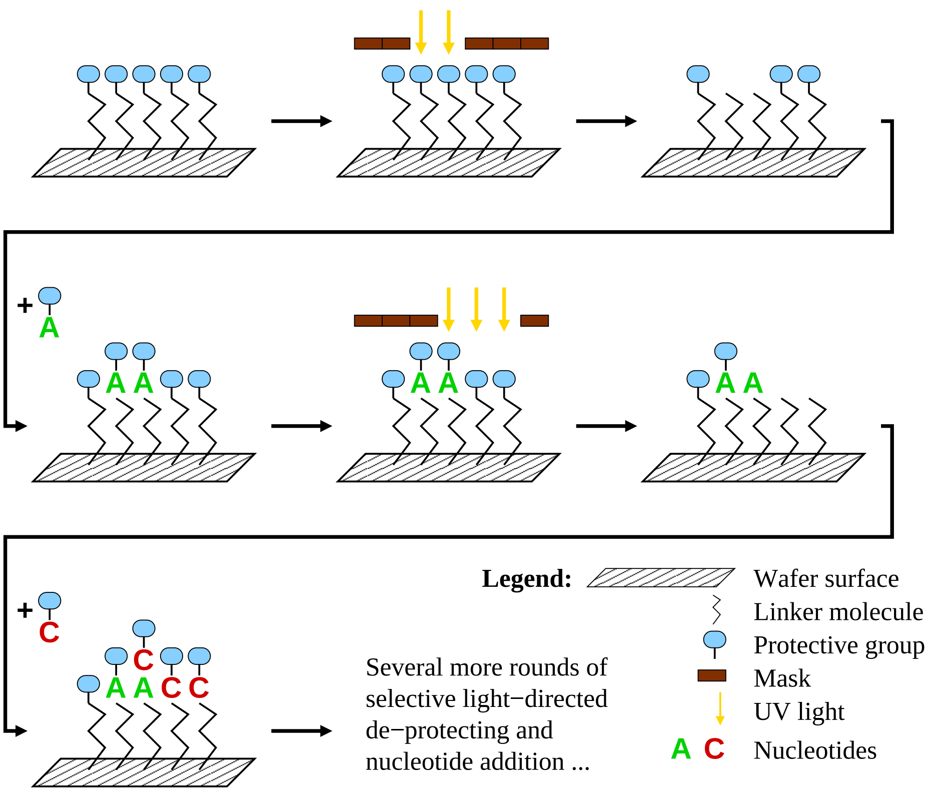


Figure 1.1: Affymetrix's probe synthesis via photolithographic masks. The chip is coated with a chemical compound and a light-sensitive protecting group; masks are used to direct light and activate selected probes for chemical coupling; nucleotides are appended to deprotected probes; the process is repeated until all probes have been fully synthesized.

1.3.2 Maskless Microarray Production

Photolithographic masks are notoriously expensive and cannot be changed once they have been manufactured. Thus, any change in the chip layout requires the production of a new set of masks. A similar method of *in situ* synthesis known as Maskless Array Synthesizer (MAS) was later developed to eliminate the need of such masks (Singh-Gasson et al., 1999). Probes are still built by repeating cycles of deprotection and chemical coupling of nucleotides. The illumination, however, relies on an array of miniature mirrors that can be independently controlled to direct or deflect the incidence of light on the chip.

NimbleGen Systems, Inc. uses its own Digital Micromirror Device (DMD) that can control up to 786 000 individual mirrors to produce microarrays with spots as small as 16 μm $\times$ 16 μm. The Geniom® system of febit biotech GmbH, a platform for customized microarray production, also uses a micromirror array to direct the synthesis process.

1.3.3 The Problem of Unintended Illumination

Regardless of which method is used to direct light (masks or micromirror arrays), it is possible that some probes are accidentally activated for chemical coupling because of light diffraction, scattering or internal reflection on the chip surface. This unwanted illumination of regions introduces unexpected nucleotides that change probe sequences, significantly reducing their chances of successful hybridization with their targets. Moreover, these faulty probes may also introduce cross-hybridizations, which can interfere in the experiments performed with the chip.

This problem is more likely to occur near the borders between a masked and an unmasked spot (in the case of maskless synthesis, between a spot that is receiving light and a spot that is not). This observation has given rise to the term *border conflict*.

It turns out that by carefully designing the *arrangement* of the probes on the chip and their *embeddings* (the sequences of masked and unmasked steps used to synthesize each probe), it is possible to reduce the risk of unintended illumination. This issue becomes even more important as there is a need to accommodate more probes on a single chip, which requires the production of spots at higher densities and, consequently, with reduced distances between probes.

In this thesis, we address the problem of designing the layout of a microarray with the goal of reducing the chances of unintended illumination, which we call Microarray Layout Problem (MLP). We use the term *layout* to refer to where and how the probes are synthesized on the chip (their arrangement and their embeddings).

Chapter 2

The Microarray Layout Problem

In this chapter we give a more precise definition of the Microarray Layout Problem (MLP) and define criteria for evaluating a given layout. The description that follows assumes that probes are synthesized with photolithographic masks, but the concepts also apply to the maskless production (with micromirror arrays). Two evaluation criteria are presented: *border length* and *conflict index*. As shown later, the conflict index model can be seen as a generalization of the border length model.

Formally, we have a set of probes $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$, where each $p_k \in \{\text{A,C,G,T}\}^*$ with $1 \leq k \leq n$ is produced by a series of T synthesis steps. Frequently, but not necessarily, all probes have the same length ℓ. Each synthesis step t uses a mask M_t to induce the addition of a particular nucleotide $N_t \in \{\text{A,C,G,T}\}$ to a subset of $\mathcal{P}$ (Figure 2.1). The *nucleotide deposition sequence* $N = N_1 N_2 \ldots N_T$ corresponding to the sequence of nucleotides added at each synthesis step is a supersequence of all $p \in \mathcal{P}$.

A microarray chip consists of a set of spots, or sites, $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$, where each spot s is specified by its coordinates on the chip surface and accommodates a unique probe $p_k \in \mathcal{P}$. Note that we usually refer to s as containing a single probe p_k although, in practice, it contains several million copies of it. Each probe is synthesized at a unique spot, hence there is a one-to-one assignment between probes and spots (if we assume that there are as many spots as probes, i.e., $m = n$). Real microarrays may have complex physical structures but we assume that the spots are arranged in a rectangular grid with n_r rows and n_c columns. We also assume that probes can be assigned to any spot.

In general, a probe can be *embedded* within N in several ways. An embedding of p_k is a T-tuple $\varepsilon_k = (\varepsilon_{k,1}, \varepsilon_{k,2}, \ldots, \varepsilon_{k,T})$ in which $\varepsilon_{k,t} = 1$ if probe p_k receives nucleotide N_t (at step t), and 0 otherwise. In particular, a *left-most embedding* is an embedding in which the bases are added as early as possible (as in ε_1 in Figure 2.1). Similarly, a *right-most embedding* is an embedding in which the bases are added as late as possible (as in ε_8 in Figure 2.1).

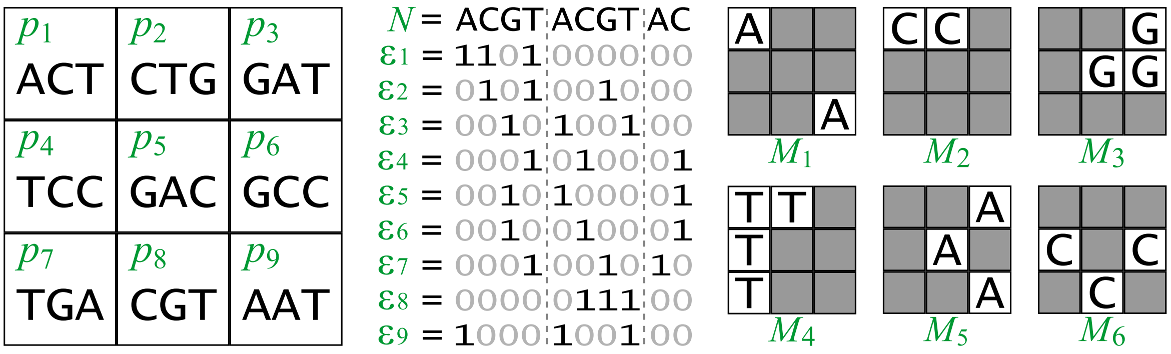


Figure 2.1: Synthesis of a hypothetical 3×3 chip with photolithographic masks. Left: chip layout and the 3-mer probe sequences. Center: deposition sequence with 2.5 cycles (cycles are delimited with dashed lines) and probe embeddings (asynchronous). Right: first six masks (masks 7 to 10 not shown).

We say that an embedding ε_k is *productive* (unmasked) at step t if $\varepsilon_{k,t} = 1$, or *unproductive* (masked) otherwise. The terms productive and unproductive can also be used to denote unmasked and masked spots, respectively.

The deposition sequence is often a repeated permutation of the alphabet, mainly because of its regular structure and because such sequences maximize the number of distinct subsequences (Chase, 1976). The deposition sequence shown in Figure 2.1 is a 2.5-time repetition of ACGT, and we thus say that it has two and a half *cycles*.

For cyclic deposition sequences, it is possible to distinguish between two types of embeddings: *synchronous* and *asynchronous*. In the former, each probe has exactly one nucleotide synthesized in every cycle of the deposition sequence; hence, 25 cycles or 100 steps are needed to synthesize probes of length 25. In the latter, probes can have any number of nucleotides synthesized in any given cycle, allowing shorter deposition sequences. For this reason, asynchronous embeddings are usually the choice for commercial microarrays. For instance, all GeneChip arrays that we know of can be asynchronously synthesized in 74 steps with 18.5 cycles of TGCA — we refer to this sequence as the *standard Affymetrix deposition sequence* (see Chapter 8).

Ideally, the deposition sequence should be as short as possible in order to reduce manufacturing time, cost and probability of errors (Rahmann, 2003). Finding the shortest deposition sequence to synthesize a set of probes is an instance of a classical computer science problem known as the Shortest Common Supersequence problem, which will be the focus of Chapter 9. For the MLP, however, we assume that N is a fixed sequence given as input.

2.1 Problem statement

Given a set of probes $\mathcal{P}$, a geometry of spots $\mathcal{S}$, and a deposition sequence N as specified above, the MLP asks to specify a chip layout (λ, ε) that consists of

1. a bijective assignment $\lambda : \mathcal{S} \to \{1, \ldots, n\}$ that specifies a probe index $k(s)$ for each spot s (meaning that $p_{k(s)}$ will be synthesized at s),
2. an assignment $\varepsilon : \{1, \ldots, n\} \to \{0,1\}^T$ specifying an embedding $\varepsilon_k = (\varepsilon_{k,1}, \ldots, \varepsilon_{k,T})$ for each probe index k, such that $N[\varepsilon_k] :\equiv (N_t)_{t:\varepsilon_{k,t}=1} = p_k$,

such that a given penalty function is minimized. We introduce two such penalty functions: total border length and total conflict index.

2.2 Border length

The first formal definition of the unintended illumination problem was given by Hannenhalli et al. (2002), who defined the *border length* $\mathcal{B}_t$ of a mask M_t as the number of borders separating masked and unmasked spots at synthesis step t, that is, the number of border conflicts in M_t. Formally,

$$\mathcal{B}_t := \frac{1}{2} \cdot \sum_{s,s' \in \mathcal{S}} \mathbb{1}_{\{s \text{ and } s' \text{ are adjacent}\}} \cdot \mathbb{1}_{\{\varepsilon_{k(s),t} \neq \varepsilon_{k(s'),t}\}}. \tag{2.1}$$

where $\mathbb{1}_{\{cond\}}$ is the indicator function that equals 1 if condition *cond* is true, and 0 otherwise. The *total border length* of a given layout (λ, ε) is the sum of border lengths over all masks, that is

$$\mathcal{B}(\lambda, \varepsilon) := \sum_{t=1}^{T} \mathcal{B}_t. \tag{2.2}$$

The *Border Length Minimization Problem* was then defined as the problem of finding a layout minimizing the total border length (Hannenhalli et al., 2002). As an example, the six masks shown in Figure 2.1 have $\mathcal{B}_1 = 4$, $\mathcal{B}_2 = 3$, $\mathcal{B}_3 = 5$, $\mathcal{B}_4 = 4$, $\mathcal{B}_5 = 8$ and $\mathcal{B}_6 = 9$. The total border length of that layout is 52 (masks M_7 to M_{10} are not shown).

Hamming distance. In the next chapters, we refer to the *Hamming distance* $H(k, k')$ between the embeddings ε_k and $\varepsilon_{k'}$ as the number of synthesis steps in which they differ. Formally,

$$H(k, k') := \sum_{t=1}^{T} \mathbb{1}_{\{\varepsilon_{k,t} \neq \varepsilon_{k',t}\}}. \tag{2.3}$$

Note that $H(k, k')$ gives the number of border conflicts generated when probes with embeddings ε_k and $\varepsilon_{k'}$ are placed in adjacent spots.

2.2.1 Lower bounds

Lower bounds for the BLMP with synchronous and asynchronous embeddings were given by Kahng et al. (2002), based on a simple graph formulation. Unfortunately, both lower bounds are not tight, and their computation is time-consuming, especially for large chips.

Synchronous embeddings. Let L be a complete directed graph over the set of probes $\mathcal{P}$ with arcs weighted with the Hamming distance between the (unique) embeddings of the corresponding probes.

Since a probe can have at most four neighbors on the chip, we delete all but the four arcs with the least weights of every node. Furthermore, assuming that the chip is a rectangular grid with n_r rows and n_c columns, we delete the heaviest $2 \cdot (n_r + n_c)$ remaining arcs, because the spots on the borders of the chip have less than four neighbors. It is not difficult to see that the cost of any placement must be greather than the total arc weight of L, and we obtain the following theorem.

Theorem 2.1. *The total arc weight of L is a lower bound on the total border length of the optimum layout with synchronous embeddings.*

Asynchronous embeddings. With asynchronous embeddings, we can construct a similar complete directed graph L'. For the arc weights, however, it is necessary to estimate the minimum number of border conflicts between the two probes (among all of their possible embeddings).

Kahng et al. (2002) observed that the number of bases of a probe p_k that can be "aligned" with bases of $p_{k'}$ cannot exceed the length of $LCS(p_k, p_{k'})$, where $LCS(p_k, p_{k'})$ is the *longest common subsequence* of p_k and $p_{k'}$. Therefore, an arc of L' between probes p_k and $p_{k'}$ can be weighted with $\ell - |LCS(p_k, p_{k'})|$, where ℓ is the length of both probe sequences (assuming probes have the same length).

We can then delete all but the four arcs with the least weights of each probe and, subsequently, the heaviest $2 \cdot (n_r + n_c)$ remaining arcs of L', to obtain the following theorem.

Theorem 2.2. *The total arc weight of L' is a lower bound on the total border length of the optimum layout with asynchronous embeddings.*

2.3 Conflict index

The border length measures the quality of an individual mask or set of masks. With this model, however, it is not possible to know how the border conflicts are distributed among the probes. Ideally, all probes should have roughly the same risk of being damaged by unintended illumination, so that all signals are affected in approximately the same way.

The *conflict index* is a quality measure defined with the aim of estimating the risk of damaging probes at a particular spot (de Carvalho Jr. and Rahmann, 2006b); it is thus a per-spot or per-probe measure instead of a per-mask measure. Additionally, it takes into account two practical considerations observed by Kahng et al. (2003a):

a) stray light might activate not only adjacent neighbors but also spots that lie as far as three cells away from the targeted spot;

b) imperfections produced in the middle of a probe are more harmful than in its extremities.

For a proposed layout (k, ε), the conflict index $\mathcal{C}(s)$ of a spot s whose probe $p_{k(s)}$ is synthesized in T masking steps according to its embedding vector $\varepsilon_{k(s)}$ is

$$\mathcal{C}(s) := \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{k(s),t}=0\}} \cdot \omega(\varepsilon_{k(s)}, t) \cdot \sum_{\substack{s'\colon \text{ neighbor} \\ \text{of } s}} \mathbb{1}_{\{\varepsilon_{k(s'),t}=1\}} \cdot \gamma(s, s') \Big). \tag{2.4}$$

The indicator functions ensure the following conflict condition: During step t, there is a conflict at spot s if and only if s is masked ($\varepsilon_{k(s),t} = 0$) and a close neighbor s' is unmasked ($\varepsilon_{k(s'),t} = 1$) — since light directed at s' may somehow reach s. When s is unmasked, it does not matter if it accidentally receives light targeted at a neighbor, and when s' is masked, there is no risk that it damages probes of s since it is not receiving light.

Function $\gamma(s, s')$ is a "closeness" measure between s and s' (to account for observation a). We defined it as

$$\gamma(s, s') := (d(s, s'))^{-2}, \tag{2.5}$$

where $d(s, s')$ is the Euclidean distance between the spots s and s'. Note that in (2.4), s' ranges over all neighboring spots that are at most three cells away from s (see Figure 2.2, left), which is in accordance with observation a.

In general, we use the terms *close neighbor* or simply *neighbor* of a spot s to refer to a spot s' that is at most three cells away (vertically and horizontally) from s. In other words, s' is inside a 7×7 region centered around s. This is in contrast to the terms *direct* or *immediate neighbor* of s, used to denote a spot s' that is adjacent to

0.06	0.08	0.10	0.11	0.10	0.08	0.06
0.08	0.13	0.20	0.25	0.20	0.13	0.08
0.10	0.20	0.50	1.00	0.50	0.20	0.10
0.11	0.25	1.00	s	1.00	0.25	0.11
0.10	0.20	0.50	1.00	0.50	0.20	0.10
0.08	0.13	0.20	0.25	0.20	0.13	0.08
0.06	0.08	0.10	0.11	0.10	0.08	0.06

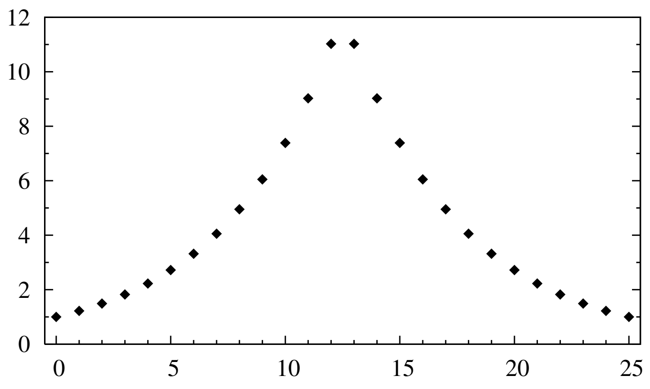


Figure 2.2: Ranges of values for both γ and ω on a typical Affymetrix chip where probes of length $\ell = 25$ are synthesized in $T = 74$ masking steps. Left: approximate values of the distance-dependent weighting function $\gamma(s, s')$ for a spot s in the center and close neighbors s'. Right: position-dependent weights $\omega(\varepsilon, t)$ on the y-axis for each value of $b_{\varepsilon,t} \in \{0, \ldots, 25\}$ on the x-axis, using $\theta = 5/\ell$ and $c = 1/\exp(\theta)$.

s (in other words, when s' shares a common border with s on the chip). Obviously, an immediate neighbor s' is also a close neighbor of s.

The position-dependent weighting function $\omega(\varepsilon, t)$ accounts for the significance of the location inside the probe where the undesired nucleotide is introduced in case of accidental illumination (observation b). We defined it as:

$$\omega(\varepsilon, t) := c \cdot \exp\left(\theta \cdot \lambda(\varepsilon, t)\right) \tag{2.6}$$

where $c > 0$ and $\theta > 0$ are constants, and for $1 \leq t \leq T$,

$$\lambda(\varepsilon, t) := 1 + \min(b_{\varepsilon,t}, \ell_\varepsilon - b_{\varepsilon,t}), \tag{2.7}$$

$$b_{\varepsilon,t} := \sum_{t'=1}^{t} \varepsilon_{t'}, \qquad \ell_\varepsilon := \sum_{t=1}^{T} \varepsilon_t = b_{\varepsilon,T}. \tag{2.8}$$

In other words, ℓ_ε is the length of the final probe specified by ε (equal to the number of ones in the embedding), and $b_{\varepsilon,t}$ denotes the number of nucleotides synthesized up to and including step t.

We can now speak of the *total conflict index* of a given layout (λ, ε) as the sum of conflict indices over all spots, that is

$$\mathcal{C}(\lambda, \varepsilon) := \sum_s \mathcal{C}(s). \tag{2.9}$$

Conflict index distance. Many of the algorithms discussed in later chapters were initially developed for border length minimization, and they usually rely on the Hamming distance defined earlier (2.3). We have adapted some of these algorithms to work with conflict index minimization by using the *conflict index distance*, which extends the Hamming distance by taking into account the position inside the probe where the conflict occurs (observation b). The conflict index distance $C(k, k')$ between the embeddings ε_k and $\varepsilon_{k'}$ is defined as:

$$C(k, k') := \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{k,t}=0 \text{ and } \varepsilon_{k',t}=1\}} \cdot \omega(\varepsilon_k, t) + \mathbb{1}_{\{\varepsilon_{k',t}=0 \text{ and } \varepsilon_{k,t}=1\}} \cdot \omega(\varepsilon_{k'}, t) \Big). \qquad (2.10)$$

The conflict index distance $C(k, k')$ can be interpreted as the sum of the conflict indices resulting from placing probes with embeddings ε_k and $\varepsilon_{k'}$ at hypothetical neighboring spots, ignoring the distance between these spots (note that there is no dependency on γ) and the conflicts generated by other neighbors.

2.3.1 The choices of γ and ω

The conflict index $\mathcal{C}(s)$ attempts to estimate the risk of damaging the probes of a spot s due to unintended illumination. The definitions of γ and ω given here are an arbitrary choice in an attempt to capture the characteristics of the problem.

However, the most appropriate choice of γ depend on several attributes of the specific technology utilized to produce the chips such as the size of the spots, the density of the probes on the chip, the physical properties of the light being used (intensity, frequency, etc.), the distance between the light source and the mask, and the distance between the mask (or the micromirrors) and the chip surface.

The most appropriate choice of ω depend on the chemical properties of the hybridization between probes and targets. Although it is generally agreed that the chances of a successful hybridization are higher if a mismatched base occurs at the extremities of the formed duplex instead of at its center (Hubbell et al., 1999), the precise effects of this position is not yet fully understood and has been an active topic of research (Binder and Preibisch, 2005).

We propose the use of an exponential function, so that ω grows exponentially from the extremities of the probe to its center (see Figure 2.2, right). The motivation behind this definition is that the probability of a successful stable hybridization of a probe with its target should increase exponentially with the absolute value of its Gibbs free energy, which increases linearly with the length of the longest perfect match between probe and target.

The parameter θ controls how steeply the exponential weighting function rises towards the middle of the probe. In our experiments, unless stated otherwise, we use probes of length $\ell = 25$, and parameters $\theta = 5/\ell$ and $c = 1/\exp(\theta)$.

Finding the best choice of γ and ω for a particular technology is beyond the scope of this thesis. We note, however, that all algorithms discussed in the next chapters were developed to work independently of the values given by these functions. In other words, should γ and ω be defined differently, no changes to the algorithms are necessary.

2.4 Chip quality measures

Most of the algorithms discussed in the next chapters can work with border length as well as conflict index minimization. In our experiments, we will usually present results with both measures, making a distinction between border length minimization (BLM) and conflict index minimization (CIM).

The relation between these two measures becomes clear if $\gamma(s, s')$ and $\omega(\varepsilon, t)$ are re-defined as follows: Set $\gamma(s, s') := 1$ if s' is a direct neighbor of s, and $:= 0$ otherwise. Also, set $c = 1/2$ and $\theta = 0$ so that $\omega(\varepsilon, t) := 1/2$ independently of the position in the probe where the conflict occurs. Now $\sum_s \mathcal{C}(s) = \sum_{t=1}^{T} \mathcal{B}_t$; that is, total border length is equivalent to the total conflict index for a particular choice of γ and ω. For the choices (2.5) and (2.6), they are not equivalent but still correlated, since a good layout has low border lengths as well as low conflict indices.

To better compare border lengths for chips of different sizes, we usually divide the total border length by the number n_b of internal borders of the chip, which equals $n_r(n_c - 1) + n_c(n_r - 1)$ if the the chip is a rectangular grid with n_r rows and n_c columns. We thus call $\mathcal{B}(\lambda, \varepsilon)/n_b$ the *normalized border length*, NBL for short, of a given layout (λ, ε). This can be further divided by the number of synthesis steps to give the *normalized border length per mask* $\mathcal{B}(\lambda, \varepsilon)/(n_b \cdot T)$. We may also refer to the normalized border length of a particular mask M_t as B_t/n_b. Since $B_t \leq n_b$, $B_t/n_b \leq 1$ and thus $\mathcal{B}(\lambda, \varepsilon)/n_b \leq T$.

Similarly, it is useful to divide the total conflict index by the number of probes on the chip, and we define the *average conflict index*, ACI for short, of a layout as $\mathcal{C}(\lambda, \varepsilon)/|\mathcal{P}|$.

2.5 How hard is the Microarray Layout Problem?

The MLP appears to be hard because of the super-exponential number of possible arrangements, although no NP-hardness proof is yet known. A formulation of the MLP as a Quadratic Assignment Problem (QAP) is given in Chapter 4. The QAP is a classical combinatorial optimization problem that is, in general, NP-hard, and particularly hard to solve in practice (Çela, 1997). Optimal solutions are thus unlikely to be found even for small chips and even if we assume that all probes have a single predefined embedding.

If we consider all possible embeddings (up to several million for a typical Affymetrix probe), the MLP is even harder. For this reason, the problem has been traditionally tackled in two phases. First, an initial embedding of the probes is fixed and an arrangement of these embeddings on the chip with minimum conflicts is sought. This is usually referred to as the *placement* phase. Second, a post-placement optimization phase *re-embeds* the probes considering their location on the chip, in such a way that the conflicts with neighboring spots are further reduced. Often, the chip is *partitioned* into smaller sub-regions before the placement phase in order to reduce running times, especially on larger chips.

The most important placement algorithms are surveyed in Chapter 3, whereas re-embedding algorithms are discussed in Chapter 5. Partitioning algorithms are the focus of Chapter 6. Finally, we present recent developments that simultaneously place and re-embed probes in Chapter 7.

Chapter 3

Placement Algorithms

The input for a placement algorithm consists of a geometry of spots $\mathcal{S}$, the deposition sequence N, and a set of probes $\mathcal{P}$, where each probe is assumed to have at least one embedding in N. The output is a one-to-one assignment λ of probes to spots. If there are more spots than probes to place, one can add enough "empty" probes that do not introduce any conflicts with the other probes (since light is never directed to their spots).

All algorithms discussed in this section assume that an initial embedding of the probes is given, which can be a left-most, right-most, synchronous or otherwise pre-computed embedding — a placement algorithm typically does not change the given embeddings.

3.1 Optimal masks for uniform arrays

Feldman and Pevzner (1994) were the first to formally address the unintended illumination problem. They showed how a placement for a *uniform array* with minimum number of border conflicts can be constructed using a two-dimensional Gray code. Uniform arrays are arrays containing all 4^ℓ probes of a given length ℓ, which require a deposition sequence of length $4 \cdot \ell$. These arrays were initially developed for the technique known as Sequencing by Hybridization (SBH).

In general, the term Gray code refers to an ordering of a set of elements in which successive elements differ in some pre-specified, usually small, way (Savage, 1997). The construction of Feldman and Pevzner is based on a two-dimensional Gray code composed of strings of length ℓ over a four-letter alphabet. It generates a $2^\ell \times 2^\ell$ array filled with ℓ-mer probes in which each pair of adjacent probes (horizontally or vertically) differs by exactly one letter. This construction is illustrated in Figure 3.1. An $(\ell + 1)$-mer array is constructed by first copying the ℓ-mer array into the upper left quadrant of the $(\ell + 1)$-mer array and reflecting it horizontally and vertically into the other three quadrants. The letter in front of the probes in the upper left quadrant

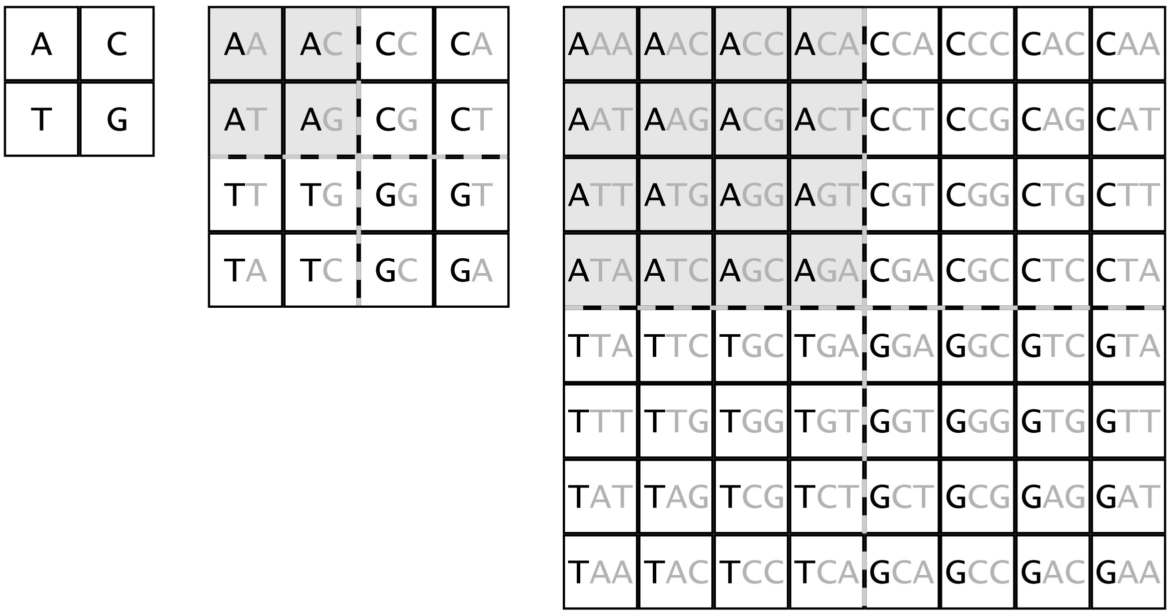

A C
T G
AA AC CC CA
AT AG CG CT
TT TG GG GT
TA TC GC GA
AAA AAC ACC ACA CCA CCC CAC CAA
AAT AAG ACG ACT CCT CCG CAG CAT
ATT ATG AGG AGT CGT CGG CTG CTT
ATA ATC AGC AGA CGA CGC CTC CTA
TTA TTC TGC TGA GGA GGC GTC GTA
TTT TTG TGG TGT GGT GGG GTG GTT
TAT TAG TCG TCT GCT GCG GAG GAT
TAA TAC TCC TCA GCA GCC GAC GAA

Figure 3.1: Construction of a placement for uniform arrays (containing the complete set of ℓ-mer probes) based on a two-dimensional Gray code, resulting in layouts with minimum number of border conflicts.

of the ℓ-mer array is added to all probes in the upper left quadrant of the $(\ell + 1)$-mer array. The probes of the other three quadrants are extended in the same way.

It can be shown that such placement generates masks with a minimum number of border conflicts if probes are synchronously embedded (see Figure 3.2). However, because this construction is restricted to uniform arrays and synchronous embeddings, it is of limited practical importance for current microarrays.

3.2 TSP and threading algorithms

The border length problem on arrays of arbitrary probes was first discussed by Hannenhalli et al. (2002). The article reports that the first Affymetrix chips were designed using a heuristic for the traveling salesman problem (TSP). The idea is to build a weighted graph with nodes representing probes, and edges containing the Hamming distances between their embeddings (see Equation 2.3). A TSP tour on this graph is heuristically constructed and *threaded* on the array in a row-by-row fashion (Figure 3.3a).

For uniform arrays, every solution of the TSP corresponds to a (one-dimensional) Gray code since consecutive elements in the tour differ in only one position, thus minimizing border conflicts between neighboring probes. For general arrays, a TSP solution also

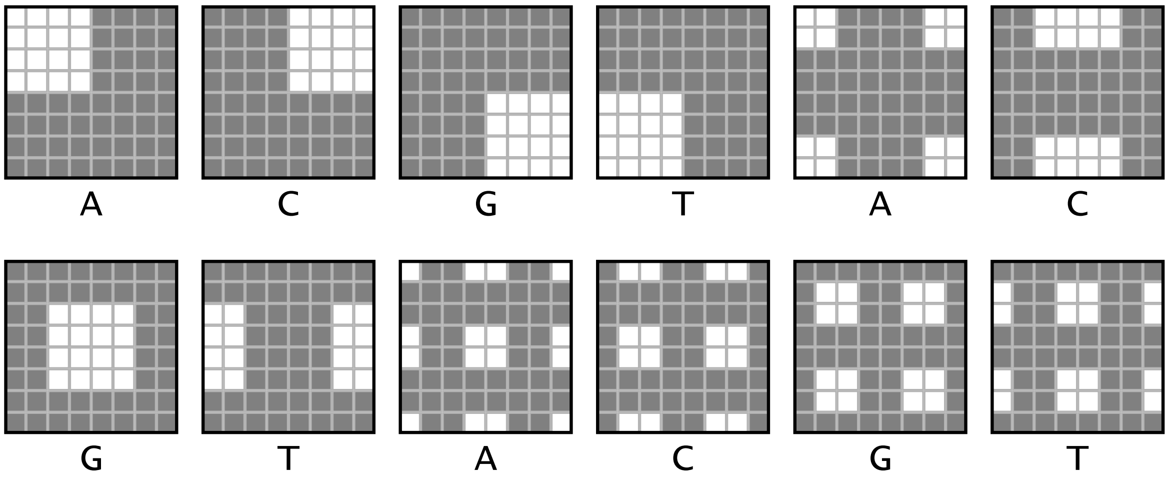


Figure 3.2: Masks for the 8×8 uniform array of Figure 3.1 when probes are synchronously embedded into $\{ACGT\}^3$. Masked spots are represented by shaded squares, unmasked spots by white squares. Note that masks of the same cycle have the same number of border conflicts.

reduces border conflicts as consecutive probes in the tour are likely to be similar. Threading the (one-dimensional) tour on a two-dimensional chip, row-by-row, on a leads to an arrangement where consecutive probes in the same row have few border conflicts, but probes in the same column may have very different embeddings.

Another problem of this approach is that the TSP is known to be NP-hard (Gross and Yellen, 2004), so computing an optimal TSP tour even for a small 300×300 array is not feasible, and only fast approximation algorithms are suitable. In practice, Hannenhalli et. al. managed to achieve marginal improvements in tour cost using the 2-opt algorithm for TSP of Lin and Kernighan (1973) and an algorithm for weighted matching due to Gabow (1976). Unfortunately, their efforts resulted in only 1.05% reduction in tour cost for a chip with 66 000 probes when compared to the greedy TSP algorithm initially used at Affymetrix.

Since improvements in the cost of the TSP tour seemed unlikely, Hannenhalli et. al. turned their attention to the problem of threading the tour on the chip. They studied several threading alternatives, which they collectively called *k-threading* (Figure 3.3). A k-threading is a variation of the standard row-by-row threading, in which the right-to-left and left-to-right paths are interspaced with alternating upward and downward movements over k sites (the row-by-row threading can be seen as a k-threading with $k = 0$); k is called the *amplitude* of the threading. Hannenhalli et. al. experimentally observed that 1-threading may reduce total border length of layouts constructed with TSP tours in up to 20% for large chips when compared to row-by-row threading.

From now on, we will use the term TSP $+k$-threading to refer to the method of computing a TSP tour and threading it on the array using k-threading.

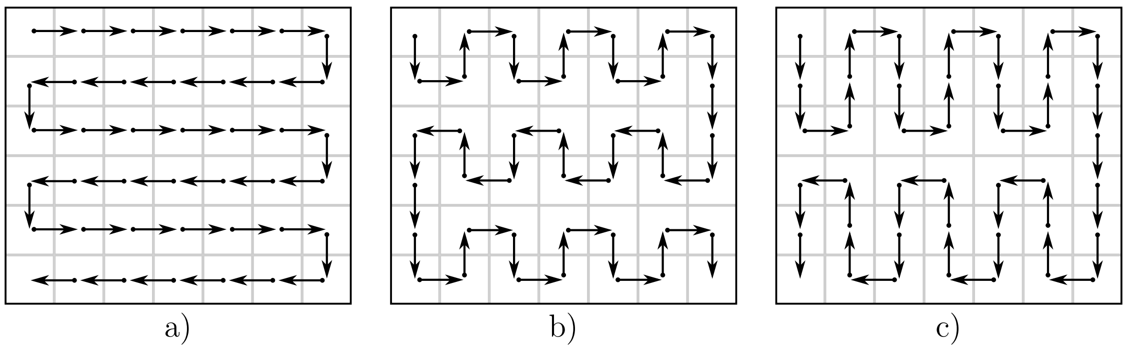


Figure 3.3: Different ways of *threading* probes on a chip. a) Standard row-by-row (0-threading); b) 1-threading; c) 2-threading.

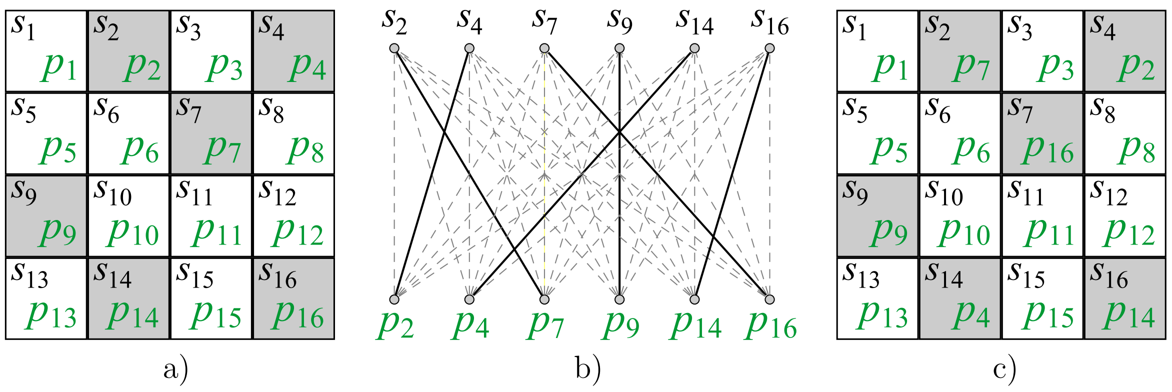
3.3 Epitaxial placement

A different strategy inspired by techniques used in the design of VLSI circuits, called Epitaxial placement, or *seeded crystal growth*, was proposed by Kahng et al. (2002). It essentially grows a placement around a single starting "seed" using a greedy heuristic. Although it was originally designed for chips with synchronous embeddings, it can be trivially implemented for asynchronous embeddings as well.

The algorithm starts by placing a random probe in the center of the array and continues to insert probes in spots adjacent to already-filled spots. Priority is given to spots whose all four neighbors are filled, in which case a probe with the minimum number of border conflicts with the neighbors is placed. Otherwise, all spots with $1 \leq i < 4$ filled neighbors are examined. For each spot s, the algorithm finds a non-assigned probe p whose number of border conflicts with the filled neighbors of s, $c(s,p)$, is minimal and assigns a normalized cost $\bar{c}(s,p) := \sigma_i \cdot c(s,p)/i$ for this assignment, where $0 < \sigma_i \leq 1$ are scaling coefficients (the authors propose $\sigma_1 = 1$, $\sigma_2 = 0.8$, and $\sigma_3 = 0.6$). The assignment with minimum $\bar{c}(s,p)$ is made and the procedure is repeated until all probes have been placed.

In order to avoid repeated cost computations, the authors propose keeping a list of probe candidates, for each spot, sorted by their normalized costs. This list must be updated whenever one of its neighbors is filled; thus, it is updated at most four times (but only two times on average).

With this algorithm, Kahng et. al. claim a further 10% reduction in border conflicts over the TSP + 1-threading approach of Hannenhalli et al. (2002). However, the Epitaxial algorithm has at least quadratic time complexity as it examines every non-placed probe to fill each spot, and large memory requirements if a list of probe candidates is kept for each spot. Hence, like the TSP approach, it does not scale well to large chips. In their experiments, the Epitaxial algorithm needed 274 seconds to design a 100×100 chip, but 4 441 seconds to design a 200×200 chip. That is a 16.2-fold



s_1 p_1 s_2 p_2 s_3 p_3 s_4 p_4
s_5 p_5 s_6 p_6 s_7 p_7 s_8 p_8
s_9 p_9 s_{10} p_{10} s_{11} p_{11} s_{12} p_{12}
s_{13} p_{13} s_{14} p_{14} s_{15} p_{15} s_{16} p_{16}
a)
s_2 s_4 s_7 s_9 s_{14} s_{16}
p_2 p_4 p_7 p_9 p_{14} p_{16}
b)
s_1 p_1 s_2 p_7 s_3 p_3 s_4 p_2
s_5 p_5 s_6 p_6 s_7 p_{16} s_8 p_8
s_9 p_9 s_{10} p_{10} s_{11} p_{11} s_{12} p_{12}
s_{13} p_{13} s_{14} p_4 s_{15} p_{15} s_{16} p_{14}
c)

Figure 3.4: Sliding-Window Matching algorithm. a) Initial arrangement of probes $p_1 \ldots p_{16}$ inside a 4×4 window (with spots $s_1 \ldots s_{16}$ and a selected maximal independent set of spots (shaded). b) Bipartite graph with selected probes and spots, and a minimum weight perfect matching (dark edges) resulting in a minimum cost re-assignment of probes to spots. c) New arrangement inside the window according to the perfect matching.

increase in running time for a 4-fold increase in number of spots. Chips of larger dimensions could not be computed because of prohibitively large running time and memory requirements.

3.4 Sliding-Window Matching

The Sliding-Window Matching algorithm (Kahng et al., 2003a), SWM for short, is not exactly a placement algorithm as it iteratively improves an existing placement that can be constructed, for instance, by TSP + 1-threading (Section 3.2).

The authors noted that the TSP tour can be conveniently substituted by lexicographically sorting the probe sequences or, alternatively, their binary embedding vectors with a linear-time radix sort. The sorting is faster, but it is also likely to produce a worse initial placement than the TSP tour, with consecutive embeddings being similar only in their first synthesis steps. The authors argue that this is of little importance in practice given that this placement is only used as a starting point for the SWM algorithm, and the lexicographical sorting should be the choice for large microarrays because computing a TSP tour takes prohibitively long for chips larger than 500×500 spots. (From now on, we will use the term sorting $+k$-threading, or simply k-threading, to refer to the method of sorting probes lexicographically and threading them on the array using k-threading.)

As its name implies, SWM works inside a window that starts at the top left of the chip and slides from left to right, top to bottom, while maintaining a certain amount of overlap between each iteration. When the window reaches the right-end of the chip, it is re-started at the left-end of the next set of rows, also retaining an overlap with the preceding set of rows.

At each iteration, the algorithm attempts to reduce the total border length inside the window by relocating some of its probes (Figure 3.4a). First, a random maximal independent set of spots is selected, and the probes assigned to these spots are removed. The term independent refers to the fact that selected spots can be re-assigned to probes without affecting the border length of other selected spots. The algorithm then creates a bipartite graph with nodes representing the removed probes and the now vacant spots (Figure 3.4b). The edges of this graph are weighted with the number of border conflicts that are generated by the corresponding assignment. Finally, a minimum weight perfect matching on this graph is computed, and the indicated assignments are made (Figure 3.4c).

A minimum weight perfect matching requires polynomial time (Gross and Yellen, 2004), but the small graphs generated by SWM can be computed rather quickly. The authors experimentally observed that the best results are obtained with small window sizes (e.g. 6×6) and an overlap of half the window size. Moreover, employing less effort in each window and executing more cycles of optimization gives better results than more effort in each window and less cycles.

Selecting an independent set of spots ensures that the cost of each new assignment can be computed independently of the other assignments. The SWM was designed for border length minimization (BLM) and it takes advantage of the fact that, in this model, an independent set of spots can be constructed by selecting spots that do not share a common border. SWM can be adapted for conflict index minimization (CIM) by using larger windows containing relatively sparse independent sets (to our knowledge, this has not been implemented yet). Therefore several random independent sets should be constructed before moving the window.

3.5 Row-Epitaxial

Row-Epitaxial (Kahng et al., 2003a) is a variant of the Epitaxial algorithm with two main differences introduced to improve scalability: i) spots are filled in a pre-defined order, namely, from top to bottom, left to right, and ii) only a limited number Q of probe candidates are considered for filling each spot.

Like SWM, Row-Epitaxial improves an initial placement that can be constructed by, for example, sorting + 1-threading. For each spot s with a probe p, it looks at the next Q probes that lie in close proximity (to the right or below s), and swaps p with the probe that generates the minimum number of border conflicts between s and its left and top neighbors.

In the experiments conducted by Kahng et al. (2003a), Row-Epitaxial was the best large-scale placement algorithm (for BLM), achieving up to 9% reduction in border conflicts over TSP + 1-threading, whereas SWM achieved slightly worse results but required significantly less time.

Row-Epitaxial can also be adapted to CIM by swapping a probe of a spot s with the probe candidate that minimizes the sum of conflict indices in a region around s restricted to those neighboring probes that are to the left or above s (those which have already found their final positions).

Table 3.1 shows the results of using Row-Epitaxial for both border length and conflict index minimization on chips with random probe sequences (uniformly generated). Probes were lexicographically sorted and left-most embedded into the standard 74-step Affymetrix deposition sequence and threaded on the array with k-threading. The resulting layouts were then used as a starting point for Row-Epitaxial.

Although Hannenhalli et al. (2002) suggested 1-threading for laying out a TSP tour on the chip, our results show that increasing the threading's amplitude from $k = 0$ to $k = 4$ usually improves the initial layout produced by sorting + k-threading, both in terms of border length and conflict index minimization. For example, increasing the amplitude from $k = 0$ to $k = 4$ reduced the normalized border length of the initial layout in up to 6.56% (from 23.6828 to 22.1279) and the average conflict index in up to 4.51% (from 689.6109 to 658.5097) on 800×800 chips.

However, the best initial layouts rarely led to the best final layout produced by Row-Epitaxial. With BLM the best results were usually achieved with $k = 0$, whereas with CIM there was no clear best value for k. In any case, the difference due to varying k for the threading were rather small for Row-Epitaxial — at most 0.78% in normalized border length (from 16.9760 with $k = 0$ to 17.1085 with $k = 4$) and 0.26% in average conflict index (from 448.0140 with $k = 0$ to 449.1653 with $k = 4$), both on a 800×800 chip with $Q = 5$K (we use "K" to denote a multiple of a thousand).

Our results also give further indication that the running time of Row-Epitaxial is approximately $O(Qn)$, i.e., linear in the chip size, where Q is a user-defined parameter that controls the number of probe candidades examined for each spot. In this way, solution quality can be traded for running time: More candidates yield better layouts but also demand more time.

3.6 Greedy

As discussed in the previous section, the best results obtained with Row-Epitaxial rarely came from the best initial layouts (produced by k-threading). This is probably because Row-Epitaxial ignores the probe order used by k-threading when it looks

Table 3.1: Normalized border length and average conflict index of layouts produced by Row-Epitaxial (Row-Eptx) on random chips of various dimensions, with initial layouts produced by sorting + k-threading. Running times are reported in minutes and include the time for k-threading and Row-Epitaxial. All results are averages over a set of five chips.

			Border length minimization			Conflict index minimization		
Dim.	Q	k	k-threading	Row-Eptx	Time	k-threading	Row-Eptx	Time
300×300	5K	0	24.9649	**18.2935**	1.1	701.8698	462.5194	4.9
		1	24.1235	18.2999	1.3	690.8091	**462.4656**	5.1
		2	23.8695	18.3072	1.2	685.5916	462.6394	4.6
		3	23.7993	18.3226	1.2	683.5980	462.5885	5.1
		4	**23.7588**	18.3279	1.3	**682.3542**	462.7775	5.1
	10K	0	24.9649	**18.1477**	2.8	701.8698	444.0354	9.7
		1	24.1235	18.1529	2.8	690.8091	444.0904	9.3
		2	23.8695	18.1519	2.9	685.5916	444.1960	10.0
		3	23.7993	18.1591	2.8	683.5980	**443.9850**	10.6
		4	**23.7588**	18.1603	2.9	**682.3542**	444.1745	9.8
	20K	0	24.9649	18.0274	7.2	701.8698	426.7824	18.9
		1	24.1235	18.0325	6.9	690.8091	426.8863	18.5
		2	23.8695	18.0277	6.6	685.5916	426.8832	19.3
		3	23.7993	**18.0272**	6.6	683.5980	426.8694	19.6
		4	**23.7588**	18.0321	7.5	**682.3542**	**426.6600**	20.2
500×500	5K	0	24.2693	**17.6000**	4.3	693.5428	456.2042	15.2
		1	23.3454	17.6095	4.1	682.2097	**456.1341**	15.2
		2	23.0797	17.6246	4.3	676.4884	456.5261	14.1
		3	22.9632	17.6474	3.8	672.8160	456.5337	14.1
		4	**22.9162**	17.6670	3.7	**671.2636**	456.8203	15.3
	10K	0	24.2693	**17.4503**	13.1	693.5428	438.7075	33.9
		1	23.3454	17.4523	12.8	682.2097	438.7379	33.6
		2	23.0797	17.4582	12.7	676.4884	**438.6477**	30.4
		3	22.9632	17.4685	12.5	672.8160	438.8183	30.8
		4	**22.9162**	17.4755	12.5	**671.2636**	438.9280	32.8
	20K	0	24.2693	17.3303	28.2	693.5428	421.1358	66.7
		1	23.3454	**17.3297**	29.0	682.2097	421.1580	63.6
		2	23.0797	17.3308	27.4	676.4884	421.1087	67.7
		3	22.9632	17.3344	27.4	672.8160	**420.9758**	65.1
		4	**22.9162**	17.3376	27.7	**671.2636**	421.0436	64.2
800×800	5K	0	23.6818	**16.9760**	12.2	689.6109	**448.0140**	36.9
		1	22.6092	16.9927	12.2	672.2254	448.1474	40.3
		2	22.3205	17.0187	11.7	664.9753	448.6130	38.6
		3	22.1958	17.0589	11.7	660.5923	448.9159	40.2
		4	**22.1279**	17.1085	12.0	**658.5097**	449.1653	40.0
	10K	0	23.6818	**16.8032**	37.0	689.6109	**432.2283**	88.2
		1	22.6092	16.8111	39.1	672.2254	432.5153	91.4
		2	22.3205	16.8235	37.7	664.9753	432.5031	85.8
		3	22.1958	16.8353	37.7	660.5923	432.6652	90.1
		4	**22.1279**	16.8622	39.0	**658.5097**	432.6980	91.9
	20K	0	23.6818	**16.6771**	83.1	689.6109	**415.6470**	174.2
		1	22.6092	16.6803	83.2	672.2254	415.7402	181.1
		2	22.3205	16.6851	83.4	664.9753	415.6622	179.7
		3	22.1958	16.6915	86.5	660.5923	415.7609	172.3
		4	**22.1279**	16.7007	80.3	**658.5097**	415.7951	190.9

for probe candidates to fill a certain spot (Row-Epitaxial always looks for candidates in the next Q spots, row-by-row, regardless of how probes were threaded on the array). Another possible disadvantage of the k-threading + Row-Epitaxial approach is that each swap made by Row-Epitaxial shuffles the probes in the not-changed spots, destroying the lexicographical order used during the threading.

In this section, we present a new placement algorithm, Greedy, that combines the Row-Epitaxial greedy heuristic and a k-threading filling strategy in a single phase, using a linked list of probes to maintain the probe order during the whole placement. Like Row-Epitaxial, Greedy fills the spots in a greedy fashion, i.e., for each spot s, it examines Q probe candidates and chooses the one that can be placed at s with minimum cost (Greedy can also be easily implemented for border length as well as for conflict index minimization).

There are two main differences to Row-Epitaxial. First, instead of (re-)filling spots row-by-row, spots are filled with k-threading (there is no need for an initial layout). Perhaps more importantly, Greedy sorts the probes lexicographically and keeps them in a doubly-linked list. This list is used to maintain the lexicographical order during placement. Moreover, it is also used to improve the chances of finding a candidate having fewer conflicts with the last placed probe (which will be its neighbor on the chip): Once a probe p is selected to fill a certain spot, it is removed from the list and the next search of candidates examines the probes around p's former position in the list, e.g., $Q/2$ probes to the left and to the right of p.

Table 3.2 shows the results of using Greedy for both border length and conflict index minimization on the same set of (random) chips that have been previously used for the experiments with Row-Epitaxial (Table 3.1). The best layouts were always achieved with $k = 0$. Interestingly, increasing the amplitude of the threading from $k = 1$ to $k = 4$ always improved the results in terms of border length. In terms of conflict index, increasing k from 1 to 3 worsened the results; in most cases, increasing it from 3 to 4 improved the results.

In terms of BLM, Greedy and Row-Epitaxial produced similar results, with the best layout of Greedy being sometimes marginally better and sometimes marginally worse than the best layout of Row-Epitaxial. In terms of CIM, however, Greedy was constantly and significantly better than Row-Epitaxial, achieving up to 5.65% reduction in average conflic index (from 415.6470 to 392.1786) on a 800×800 chip with $Q = 20$K.

In our results, Greedy was between 13.9% and 59.9% slower than Row-epitaxial in the BLM case (19.7% on average), and between 3.7% and 18.1% in the CIM case (only 5.6% on average). The difference between Row-epitaxial and Greedy drops in the CIM case because the extra time spent in computing the cost of each candidate is higher than in the BLM case, which reduces the impact of the time required to keep the doubly-linked list.

Table 3.2: Normalized border length (NBL) and average conflict index (ACI) of layouts produced by Greedy on random chips of various dimensions. The results of Row-Epitaxial on the same set of chips (Table 3.1) is shown for comparison. Running times in minutes.

			Border length minimization				Conflict index minimization			
			Row-Epitaxial		Greedy		Row-Epitaxial		Greedy	
Dim.	Q	k	NBL	Time	NBL	Time	ACI	Time	ACI	Time
300^2	5K	0	**18.2935**	1.1	**18.3182**	1.6	462.5194	4.9	**440.5166**	5.4
		1	18.2999	1.3	18.5037	1.6	**462.4656**	5.1	444.7837	5.3
		2	18.3072	1.2	18.4222	1.6	462.6394	4.6	446.8662	5.3
		3	18.3226	1.2	18.3863	1.6	462.5885	5.1	447.7464	5.0
		4	18.3279	1.3	18.3728	1.5	462.7775	5.1	447.6559	5.3
	10K	0	**18.1477**	2.8	**18.1830**	4.3	444.0354	9.7	**426.3480**	10.9
		1	18.1529	2.8	18.3912	4.7	444.0904	9.3	429.5617	11.3
		2	18.1519	2.9	18.3058	4.5	444.1960	10.0	431.7555	11.1
		3	18.1591	2.8	18.2732	4.6	**443.9850**	10.6	432.6821	11.3
		4	18.1603	2.9	18.2415	4.6	444.1745	9.8	432.3800	11.0
	20K	0	18.0274	7.2	**18.0576**	9.6	426.7824	18.9	**415.5003**	30.3
		1	18.0325	6.9	18.2813	9.2	426.8863	18.5	418.2357	21.3
		2	18.0277	6.6	18.1985	9.2	426.8832	19.3	419.4866	21.1
		3	**18.0272**	6.6	18.1617	9.5	426.8694	19.6	420.7345	20.1
		4	18.0321	7.5	18.1328	8.8	**426.6600**	20.2	420.7332	21.0
500^2	5K	0	**17.6000**	4.3	**17.5830**	5.8	456.2042	15.2	**432.3023**	15.9
		1	17.6095	4.1	17.7842	5.3	**456.1341**	15.2	437.2417	16.2
		2	17.6246	4.3	17.7087	5.3	456.5261	14.1	439.7432	15.6
		3	17.6474	3.8	17.6759	5.4	456.5337	14.1	441.3441	16.2
		4	17.6670	3.7	17.6561	5.4	456.8203	15.3	441.0668	16.1
	10K	0	**17.4503**	13.1	**17.4673**	15.8	438.7075	33.9	**415.6951**	35.4
		1	17.4523	12.8	17.6765	16.0	438.7379	33.6	419.7788	33.4
		2	17.4582	12.7	17.5936	16.8	**438.6477**	30.4	422.1943	36.3
		3	17.4685	12.5	17.5550	16.2	438.8183	30.8	424.0554	34.6
		4	17.4755	12.5	17.5324	15.7	438.9280	32.8	423.7936	35.2
	20K	0	17.3303	28.2	**17.3554**	33.3	421.1358	66.7	**401.4609**	67.1
		1	**17.3297**	29.0	17.5829	34.0	421.1580	63.6	404.9949	69.8
		2	17.3308	27.4	17.4939	34.1	421.1087	67.7	406.9576	67.8
		3	17.3344	27.4	17.4519	34.7	**420.9758**	65.1	408.5048	69.4
		4	17.3376	27.7	17.4273	33.7	421.0436	64.2	408.4556	68.4
800^2	5K	0	**16.9760**	12.2	**16.9124**	15.3	**448.0140**	36.9	**426.0757**	41.9
		1	16.9927	12.2	17.1259	15.5	448.1474	40.3	430.9759	42.7
		2	17.0187	11.7	17.0551	15.1	448.6130	38.6	433.8504	42.9
		3	17.0589	11.7	17.0214	15.4	448.9159	40.2	435.4797	43.3
		4	17.1085	12.0	17.0009	15.4	449.1653	40.0	435.2589	43.2
	10K	0	**16.8032**	37.0	**16.7951**	45.5	**432.2283**	88.2	**408.3982**	91.4
		1	16.8111	39.1	17.0122	43.2	432.5153	91.4	412.9971	95.1
		2	16.8235	37.7	16.9333	45.1	432.5031	85.8	415.7934	91.7
		3	16.8353	37.7	16.8935	45.9	432.6652	90.1	417.5229	95.2
		4	16.8622	39.0	16.8748	45.6	432.6980	91.9	417.5098	91.3
	20K	0	**16.6771**	83.1	**16.6980**	95.8	**415.6470**	174.2	**392.1786**	186.4
		1	16.6803	83.2	16.9263	93.2	415.7402	181.1	396.3923	185.9
		2	16.6851	83.4	16.8376	93.8	415.6622	179.7	399.0043	186.9
		3	16.6915	86.5	16.7947	94.4	415.7609	172.3	400.7189	183.3
		4	16.7007	80.3	16.7727	97.2	415.7951	190.9	400.7257	189.3

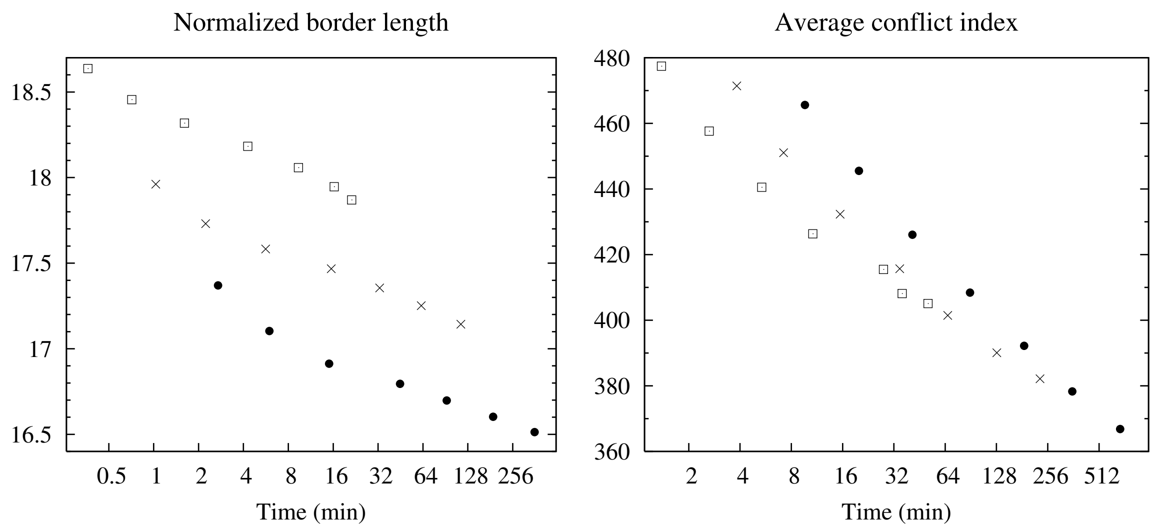


Figure 3.5: Trade-off between solution quality and running time (in logarithmic scale) with the Greedy algorithm on random chips of dimensions 300×300 (▫), 500×500 ($\times$) and 800×800 ($\bullet$). The number Q of candidates per spot are 1.25K, 2.5K, 5K, 10K, 20K, 40K, and 80K (from left to right). Layouts are measured by normalized border length (left) and average conflict index (right).

It should be noted that, like Row-Epitaxial, Greedy has the drawback of treating the last spots of a chip "unfairly": While Q probe candidades are examined for each of the first $n - Q + 1$ filled spots, the last $Q - 1$ spots have fewer than Q candidates (in particular, when the last spot is being filled, there is only one probe candidate). As a result, we usually observe comparatively higher levels of conflicts in the last filled spots.

We also observed that, in terms of border length, increasing Q above 5K has little positive effect (see Figure 3.5). For instance, on 800×800 chips, increasing Q from 5K to 20K reduced the normalized border length by only 1.27% (from 16.9124 to 16.6980 with $k = 0$), while requiring approximately six times more time. In terms of conflict index, however, increasing Q even above 40K still results in significant improvements for large chips. For instance, on 800×800 chips, increasing Q from 40K to 80K reduced the average conflict index by 3.18% (from 378.3110 to 366.8446 with $k = 0$, data not shown).

The fact that increasing Q has more effect in terms of conflict index is probably because, in this measure, there is more room for optimization as the conflicts can be moved to the extremities of the probes (while retaining the same number of border conflicts) and a larger number of neighbors are involved.

Figure 3.6 shows the normalized border length per masking step of layouts produced by Greedy for a 500×500 chip with border length and conflict index minimization.

With BLM, the generated layout has most border conflicts concentrated between steps 7 and 58. The last masks of this layout have low levels of border conflicts because the probes are left-most embedded, which leaves most embeddings in an unproductive state during the final synthesis steps. As a result of the lexicographical sorting of probes, the first masks also have relatively few conflicts. A representation of selected photolithographic masks for this layout are shown in Figure 3.7. Layers of masked and unmasked regions that result from sorting probes lexicographically can be seen in masks $M_1 \dots M_8$. Masks $M_9 \dots M_{62}$ are very "noisy" as there seems to be little regularity in their arrangement. From M_{62} on, masks start to get "darker" as most probes have been already fully synthesized.

With CIM, Greedy shifts border conflicts away from the steps that synthesize the probes' middle bases (between steps 20 and 45; see Figure 3.6), which effectively reduces the average conflict index. Not surprisingly, this reduction comes at the expense of an increase in total border length — the normalized border length of this particular chip rose from 17.3513 with BLM to 19.8461 with CIM. Figure 3.8 shows a representation of selected masks for this layout. Note that the first masks ($M_1 \dots M_4$) still exhibit a "layered" structured, althought the layers are much narrower and the masks noisier than the first masks of Figure 3.7. In the middle masks $M_{20} \dots M_{45}$, especially between M_{25} and M_{40}, it is possible to see clusters of masked and unmasked spots that cause the reduction in average conflict index.

3.7 Summary

In this chapter, we have surveyed placement algorithms for the Microarray Layout Problem, including an optimal placement strategy for uniform arrays based on a two-dimensional Gray code. For general arrays, we have presented more experimental results with Row-Epitaxial, the best known placement algorithm to date, and studied the impact of the choice of threading for its initial layout.

We have also introduced a new placement algorithm, Greedy. Greedy achieved similar results in terms of BLM and better results in terms of CIM compared to Row-Epitaxial. For BLM, Row-Epitaxial is faster than Greedy and should still be the method of choice. For CIM, however, we believe that the improvements achieved by Greedy over Row-Epitaxial justify the small increase in running time.

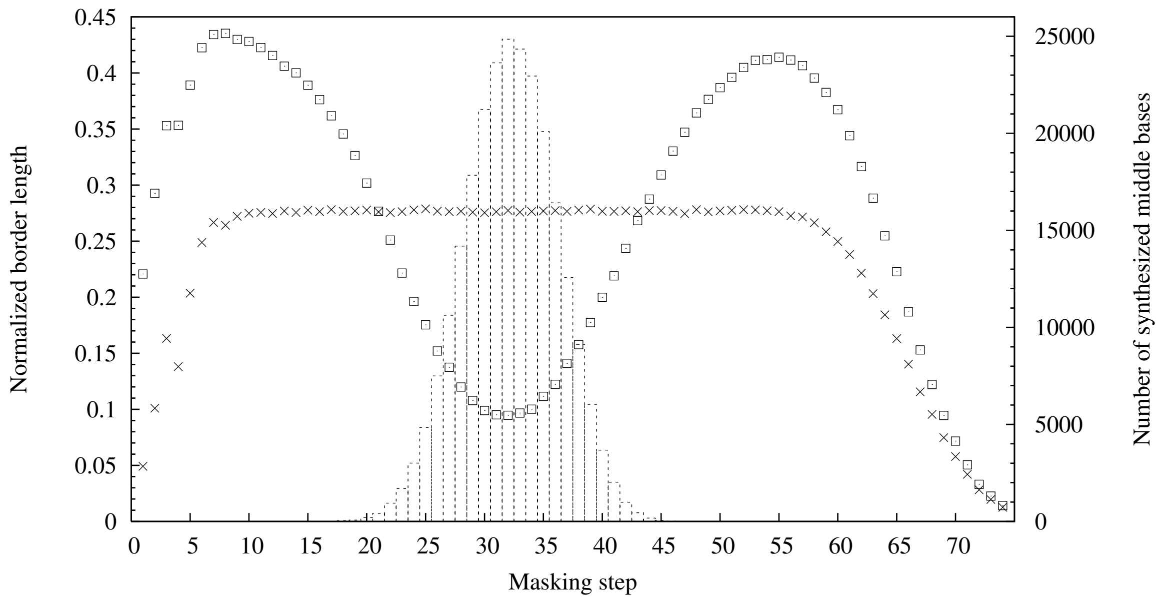


Figure 3.6: Normalized border length (on the left y-axis) per masking step of layouts produced by Greedy for a 500 × 500 chip with border length (×) and conflict index (⊡) minimization using 0-threading and $Q = 20K$. Chip contained random probe sequences left-most embedded in the standard 74-step Affymetrix deposition sequence. The number of middle bases synthesized at each step is shown in boxes (right y-axis).

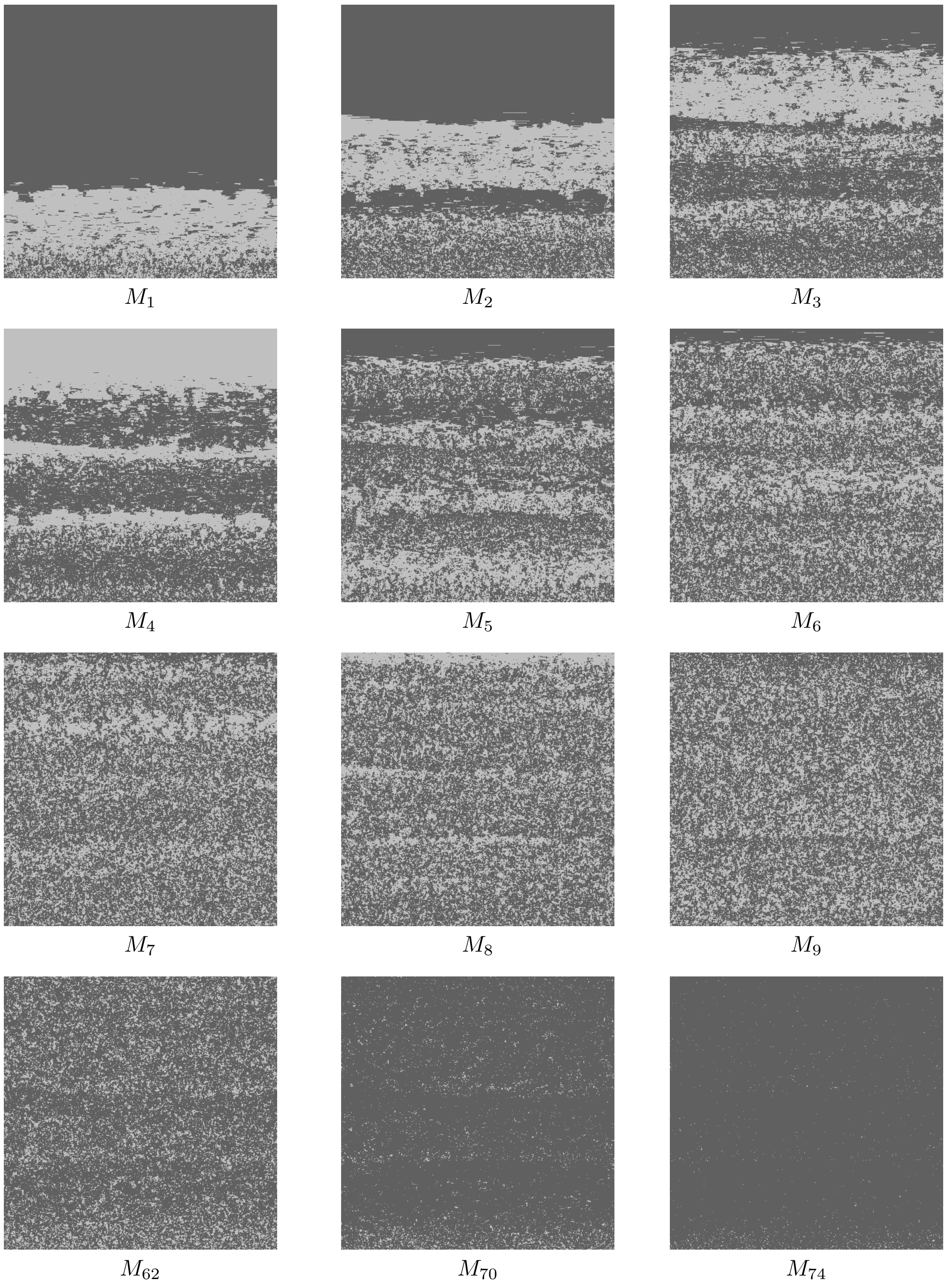


Figure 3.7: Selected masks generated by Greedy with border length minimization for a random 500×500 chip with 25-mer probes leftmost embedded in the standard Affymetrix deposition sequence. Unmasked (masked) spots are represented by light (dark) dots.

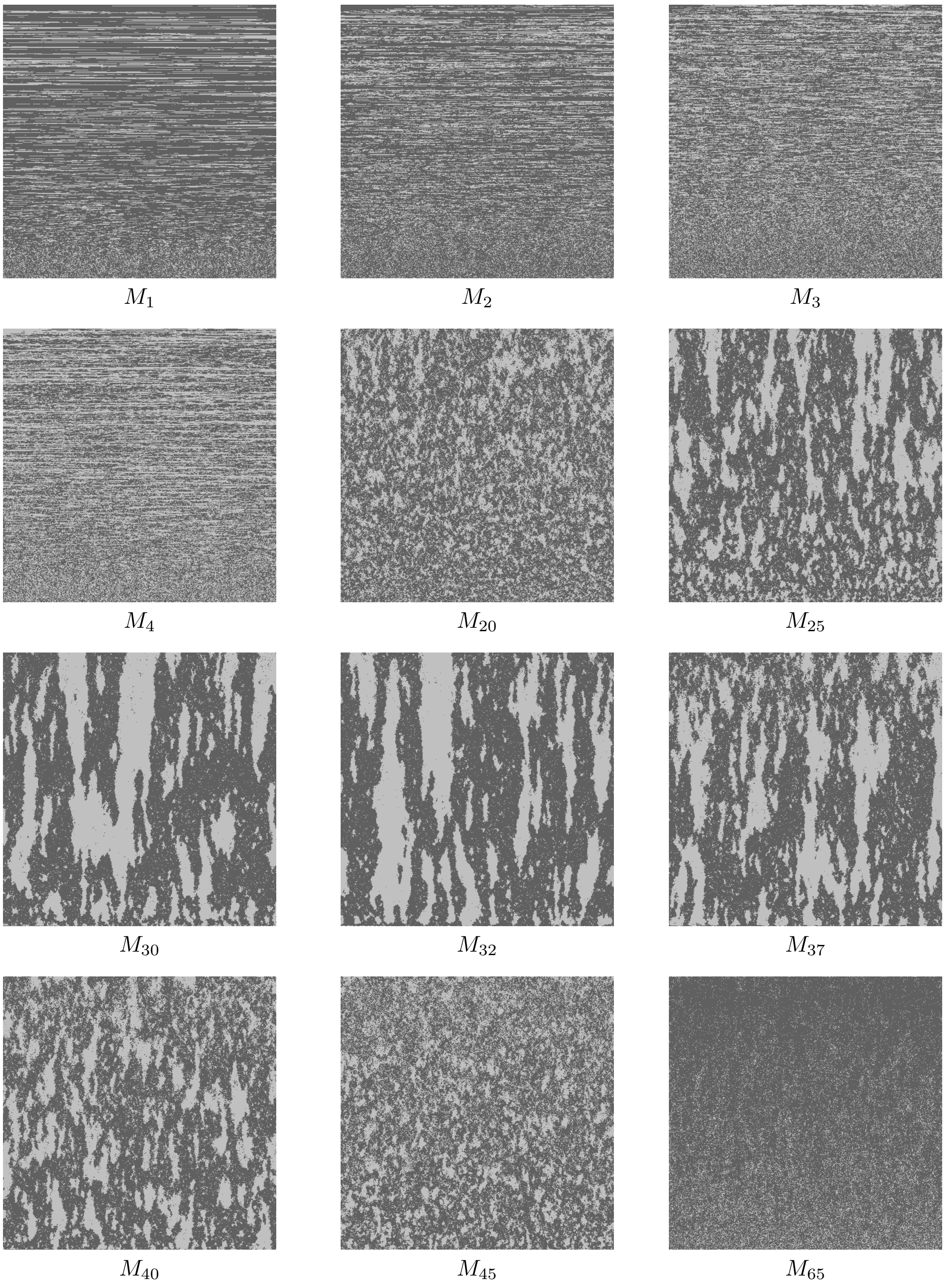


Figure 3.8: Selected masks generated by Greedy with conflict index minimization for a random 500×500 chip with 25-mer probes leftmost embedded in the standard Affymetrix deposition sequence. Unmasked (masked) spots are represented by light (dark) dots.

Chapter 4

MLP and the Quadratic Assignment Problem

In this chapter, we show that the Microarray Placement Problem (MLP) with general distance-dependent and position-dependent weights is an instance of the *Quadratic Assignment Problem* (QAP), a classical combinatorial optimization problem introduced by Koopmans and Beckmann (1957), which opens up the way for using QAP techniques to design microarray chips.

We then use an existing QAP heuristic algorithm called GRASP to design the layout of small artificial chips, comparing our results with the best known placement algorithm. The chapter ends with a discussion about how this approach can be combined with other existing algorithms to design and improve larger microarrays.

4.1 Quadratic assignment problem

The quadratic assignment problem (QAP) can be stated as follows. Given $n \times n$ real-valued matrices $F = (f_{ij}) \geq 0$ and $D = (d_{kl}) \geq 0$, find a permutation π of $\{1, 2, \dots n\}$ such that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \to \min . \tag{4.1}$$

The attribute *quadratic* stems from the fact that the target function can be written with n^2 binary indicator variables $x_{ik} \in \{0, 1\}$, where $x_{ik} := 1$ if and only if $k = \pi(i)$. The objective (4.1) then becomes

$$\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot \sum_{k=1}^{n} \sum_{l=1}^{n} d_{kl} \cdot x_{ik} \cdot x_{jl} \to \min,$$

such that $\sum_k x_{ik} = 1$ for all i, $\sum_i x_{ik} = 1$ for all k, and $x_{ik} \in \{0, 1\}$ for all (i, k). The objective function is a quadratic form in x.

The QAP has been used to model a variety of real-life problems. One common example is the facility location problem where n facilities must be assigned to n locations. The facilities could be, for instance, the clinics, doctors or services (X-ray, emergency room, etc.) provided by a hospital and the locations could be the available rooms of the hospital building.

In this scenario, F is called the *flow matrix* as f_{ij} represents the flow of materials or persons from facility i to facility j. Matrix D is called the *distance matrix*, as d_{kl} gives the distance between locations k and l. One unit of flow is assumed to have an associated cost proportional to the distance between the facilities, and the optimal permutation π defines a one-to-one assignment of facilities to locations with minimum cost.

4.2 QAP formulation of the MLP

The MLP can be seen as an instance of the QAP, where we want to find a one-to-one correspondence between spots and probes minimizing a given penalty funtion such as total border length or total conflict index (defined in Chapter 2). To formulate it, we use the facility location example by viewing the probes as locations and the spots as facilities, i.e., the spots are assigned to the probes. The flow matrix F then contains the “closeness” values between spots, while the distance matrix D contains the conflicts between probe embeddings.

We first give the general formulation for conflict index minimization case; the border length minimization case is obtained by using the particular weight functions given in Section 2.4.

In a realistic setting, we may have more spots available than probes to place. Below, we show that this does not cause problems as we can add enough “empty” probes and define their weights appropriately.

Perhaps more severely, we assume that all probes have a single pre-defined embedding in order to force a one-to-one relationship. A more elaborate formulation would consider all possible embeddings of a probe, but then it becomes necessary to ensure that only one embedding of each probe is used. This still leads to a quadratic integer programming problem, albeit with slightly different side conditions.

Our goal is to design a microarray minimizing the sum of conflict indices over all spots $s \in \mathcal{S}$, i.e.,

$$\sum_{s \in \mathcal{S}} \mathcal{C}(s) \rightarrow \min .$$

The "flow" f_{ij} between spots i and j depends on their distance on the chip; in accordance with the conflict index model, we set

$$f_{ij} := \mathbb{1}_{\{i,j \text{ neighbors}\}} \cdot \gamma(i,j) \tag{4.2}$$

where "neighbors" means that spots i and j are at most three cells away (horizontally and vertically) from each other. Note that most of the flow values on large arrays are zero. For border length minimization, the case is even simpler: We set $f_{ij} := 1$ if spots i and j are adjacent, and $f_{ij} := 0$ otherwise.

The "distance" d_{kl} between probes k and l depends on the conflicts between their embeddings ε_k and ε_l. To account for possible "empty" probes to fill up surplus spots, we set $d_{kl} := 0$ if k or l or both refer to an empty probe — i.e., empty probes never contribute to the target function since we do not mind if nucleotides are erroneously synthesized on spots assigned to empty probes. For real probes, we set

$$d_{kl} := \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{k,t}=0\}} \cdot \omega(\varepsilon_k, t) \cdot \mathbb{1}_{\{\varepsilon_{l,t}=1\}} \Big). \tag{4.3}$$

Note that d_{kl} is related to the conflict index distance $C(k,l)$ defined in Section 2.3 (Equation 2.10):

$$\begin{aligned}
& d_{kl} + d_{lk} \\
= \; & \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{k,t}=0\}} \cdot \omega(\varepsilon_k, t) \cdot \mathbb{1}_{\{\varepsilon_{l,t}=1\}} \Big) + \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{l,t}=0\}} \cdot \omega(\varepsilon_l, t) \cdot \mathbb{1}_{\{\varepsilon_{k,t}=1\}} \Big) \\
= \; & \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{k,t}=0 \text{ and } \varepsilon_{l,t}=1\}} \cdot \omega(\varepsilon_k, t) \Big) + \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{l,t}=0 \text{ and } \varepsilon_{k,t}=1\}} \cdot \omega(\varepsilon_l, t) \Big) \\
= \; & \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{k,t}=0 \text{ and } \varepsilon_{l,t}=1\}} \cdot \omega(\varepsilon_k, t) + \mathbb{1}_{\{\varepsilon_{l,t}=0 \text{ and } \varepsilon_{k,t}=1\}} \cdot \omega(\varepsilon_l, t) \Big) \\
= \; & C(k,l)
\end{aligned}$$

In the case of border length minimization, where $\theta = 0$ and $c = 1/2$ (see Section 2.4), we obtain that $d_{kl} + d_{lk} = H(k,l) = H(l,k)$, where H_{kl} denotes the Hamming distance between the embeddings ε_k and ε_l (Equation 2.3).

It now follows that for a given assignment π, we have,

$$f_{ij} \cdot d_{\pi(i)\pi(j)} = \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{\pi(i),t}=0\}} \cdot \omega(\varepsilon_{\pi(i)}, t) \cdot \mathbb{1}_{\{\varepsilon_{\pi(j),t}=1\}} \cdot \mathbb{1}_{\{i,j \text{ neighbors}\}} \cdot \gamma(i,j) \Big).$$

The objective function (4.1) then becomes

$$\begin{aligned}
& \sum_i \sum_j f_{ij} \cdot d_{\pi(i)\pi(j)} \\
= \; & \sum_i \sum_j \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{\pi(i),t}=0\}} \cdot \omega(\varepsilon_{\pi(i)}, t) \cdot \mathbb{1}_{\{\varepsilon_{\pi(j),t}=1\}} \cdot \mathbb{1}_{\{i,j \text{ neighbors}\}} \cdot \gamma(i,j) \Big) \\
= \; & \sum_i \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{\pi(i),t}=0\}} \cdot \omega(\varepsilon_{\pi(i)}, t) \cdot \sum_j \mathbb{1}_{\{i,j \text{ neighbors}\}} \cdot \mathbb{1}_{\{\varepsilon_{\pi(j),t}=1\}} \cdot \gamma(i,j) \Big) \\
= \; & \sum_i \sum_{t=1}^{T} \Big(\mathbb{1}_{\{\varepsilon_{\pi(i),t}=0\}} \cdot \omega(\varepsilon_{\pi(i)}, t) \cdot \sum_{\substack{j:\text{ neighbor} \\ \text{of } i}} \mathbb{1}_{\{\varepsilon_{\pi(j),t}=1\}} \cdot \gamma(i,j) \Big) \\
= \; & \sum_i \mathcal{C}(i),
\end{aligned}$$

and indeed equals the total conflict index with our definitions of $F = (f_{ij})$ and $D = (d_{kl})$.

Remark. Note that it is technically possible to switch the definitions of F and D, i.e., to assign probes to spots instead of spots to probes as we do now, without modifying the mathematical problem formulation. However, this would lead to high distance values for neighboring spots and many zero distance values for independent spots, a somewhat counterintuitive model. Also, some QAP heuristics initially find pairs of objects with large flow values and place them close to each other. Therefore, the way of modeling F and D may be significant.

4.3 QAP heuristics

We have shown how the microarray placement problem can be modeled as a quadratic assignment problem. However, the QAP is known to be NP-hard and particularly hard to solve in practice. Instances of size larger than $n = 20$ are generally considered to be impossible to solve to optimality. Fortunately, several heuristics exist, including approaches based on tabu search, simulated annealing and genetic algorithms (for a survey, see Çela, 1997). Our formulation is thus of interest because we can now use existing QAP heuristics to design the layout of microarrays minimizing either the sum of border lengths or conflict indices.

As an example, we briefly describe a general QAP heuristic known as GRASP (Li et al., 1994), which was first used for solving the QAP by Feo and Resende (1995), and an improved version called GRASP with path-relinking (Oliveira et al., 2004), that we used to design small microarray chips with our formulation.

4.3.1 GRASP with Path-relinking

GRASP (Greedy Randomized Adaptive Search Procedure) is comprised of two phases: a construction phase where a random feasible solution is built, and a local search phase where a local optimum in the neighborhood of that solution is sought. In the following description we use the terms of the facility location problem: f_{ij} is the flow between facilities i and j, d_{kl} is the distance between locations k and l.

The construction phase starts by sorting the $(n^2 - n)$ elements of the distance matrix in increasing order and keeping the smallest $E := \lfloor \beta(n^2 - n) \rfloor$ elements, where $0 < \beta < 1$ is a restriction parameter given as input.

$$d_{k_1 l_1} \leq d_{k_2 l_2} \leq \cdots \leq d_{k_E l_E}.$$

Similarly, the $(n^2 - n)$ elements of the flow matrix are sorted, this time in decreasing order, and the largest E elements are kept:

$$f_{i_1 j_1} \geq f_{i_2 j_2} \geq \cdots \geq f_{i_E j_E}.$$

Then, the costs of assigning pairs of facilities to pairs of locations are computed. The cost of initially assigning facility i_q to location k_q and facility j_q to location l_q for some $q \in \{1, \ldots, E\}$ is $d_{k_q l_q} f_{i_q j_q}$. GRASP sorts the vector

$$(d_{k_1 l_1} f_{i_1 j_1},\ d_{k_2 l_2} f_{i_2 j_2},\ \ldots,\ d_{k_E l_E} f_{i_E j_E}),$$

keeping the $\lfloor \alpha E \rfloor$ smallest elements, where $0 < \alpha < 1$ is another restriction parameter. A simultaneous assignment of a pair of facilities to a pair of locations is selected at random among those with the $\lfloor \alpha E \rfloor$ smallest costs, and a feasible solution is then built by making a series of greedy assignments.

In the local search phase, GRASP searches for a local optimum in the neighborhood of the constructed solution. Several search strategies and definitions of neighborhood can be used. One possible approach is to check every possible swap of assignments and make only those which improve the current solution until no further improvements can be made.

The construction and local search phases are repeated for a given number of times, and the best solution found is returned.

Path-relinking. GRASP takes no advantage of the knowledge gained in previous iterations to build or improve an obtained solution, i.e., each new solution is built from scratch.

GRASP with path-relinking is an extension of the basic GRASP algorithm that uses an "elite set" to store the best solutions found. It incorporates a third phase that chooses, at random, one elite solution that is used to improve the solution produced at the end of the local search phase.

Solutions p and q are combined as follows. For every location $k = 1, \ldots, n$, the path-relinking algorithm attempts to exchange facility p_k assigned to location k in solution p with facility q_k assigned to location k in the elite solution. In order to keep the solution p feasible, it exchanges p_k with p_l, where $p_l = q_k$. This exchange is performed only if it results in a better solution. The result of the path-relinking phase is a solution r that is at least as good as the better of p and q.

4.4 Results

We present experimental results of using GRASP with path-relinking (GRASP-PR) for designing the layout of small artificial chips, and compare them with the layouts produced by the Greedy placement algorithm (described in Section 3.6), with the number Q of candidates per spot set to a sufficiently large value so that all available probes are considered for each spot.

We used a C implementation of GRASP-PR provided by Oliveira et al. (2004) with default parameters (32 iterations, $\alpha = 0.1$, $\beta = 0.5$, and elite set of size 10). The main routine takes three arguments: the dimension n of the problem (in our case, the number of spots or probes) and matrices F and D. The matrices were generated using the formulations presented in Section 4.2.

The data set consists of chips with probes of length 25 uniformly generated and asynchronously embedded in a deposition sequence of length 74. The running times and the border lengths of the resulting layouts are shown in Table 4.1 (all results are averages over a set of ten chips).

Our results show that GRASP-PR produces layouts with lower border lengths than Greedy on the smaller chips. On 6×6 chips, GRASP-PR outperforms Greedy by 2.14 percentage points on average ($15.94\% - 13.80\%$), when compared to the initial random layout. On 9×9 chips, however, this difference drops to 0.16 percentage points, while Greedy generates better layouts on 11×11 or larger chips. In terms of running time, Greedy is faster and shows little variation as the number of probes grows. In contrast, the time required to compute a layout with GRASP-PR increases at a fast rate.

Table 4.1: Total border length of random chips compared with the layouts produced by Greedy and GRASP with path-relinking. Reductions in border length are reported in percentages compared to the random layout.

	Random	Greedy placement			GRASP with path-relinking		
Chip dimension	Border length	Border length	Reduction (%)	Time (sec.)	Border length	Reduction (%)	Time (sec.)
6×6	1 989.20	1 714.60	13.80	0.01	1 672.20	15.94	2.73
7×7	2 783.20	2 354.60	15.40	0.02	2 332.60	16.19	6.43
8×8	3 721.20	3 123.80	16.05	0.03	3 099.13	16.72	12.49
9×9	4 762.00	3 974.80	16.53	0.05	3 967.20	16.69	25.96
10×10	5 985.20	4 895.60	18.20	0.06	4 911.40	17.94	47.57
11×11	7 288.40	5 954.40	18.30	0.10	5 990.73	17.80	87.48
12×12	8 714.00	7 086.20	18.68	0.11	7 159.80	17.84	152.42

Table 4.2: Average conflict indices of random chips compared with the layouts produced by Greedy and GRASP with path-relinking.

	Random	Greedy placement			GRASP with path-relinking		
Chip dimension	Avg. C. Index	Avg. C. Index	Reduction (%)	Time (sec.)	Avg. C. Index	Reduction (%)	Time (sec.)
6×6	524.28	495.15	5.56	0.05	467.08	10.91	3.68
7×7	558.25	521.90	6.51	0.07	489.32	12.35	8.84
8×8	590.51	551.84	6.55	0.09	515.69	12.67	19.48
9×9	613.25	568.62	7.28	0.11	533.79	12.96	38.83
10×10	628.50	576.49	8.28	0.11	539.69	14.13	73.09
11×11	642.72	588.91	8.37	0.12	551.41	14.21	145.67
12×12	656.86	598.21	8.93	0.12	561.21	14.56	249.19

Table 4.2 shows better results in terms of conflict indices. GRASP-PR consistently produces better layouts on all chip dimensions, achieving up to 6.38% less conflicts on 10×10 chips, for example, when compared to Greedy. In terms of running times, GRASP-PR is even slower than in the border length case. The reason is not clear, but it could be because the distance matrix contains fewer zero entries with the conflict index formulation.

The gains in terms of conflict index of both Greedy and GRASP-PR are clearly less than the gains in terms of border length (when compared to the initial random layout). This may be because the probe embeddings are fixed and the reduction of conflicts is restricted to the relocation of the probes, which only accounts for one part of the conflict index model.

4.5 Discussion

The QAP is notoriously hard to solve, and currently known exact methods start to take prohibitively long already for slightly more than 20 objects, i.e., we could barely solve the problem exactly for 5×5 arrays. Fortunately, the literature on QAP heuristics is rich, as many problems in operations research can be modeled as QAPs. Here we used one such heuristic to identify the potential of the MLP-QAP-relation.

As our results show, however, even heuristic algorithms are too slow to deal with chips of dimensions larger than 12×12, and although we could design a 20×20 chip with a QAP heuristic within a day, we have to keep in mind that this would still be a very small part of bigger problem as real microarray dimensions range from 200×200 up to 1164×1164.

For this reason, we restricted our experiments to such small chips and QAP heuristics that could handle the problems within a few minutes. Up to now, finding exact solutions even to these small microarrays seems to be an incredible hard task. We mention here experiments conducted by Dr. Peter Hahn, who used two branch-and-bound algorithms to solve some problem instances from Table 4.1. With RTL-2 (Adams et al., To appear), it was possible to find two solutions with total border length of 1 652 for a selected 6×6 chip, being only 1.43% better than the solution found with GRASP-PR (1 676), although it took RTL-2 about 6.5 hours, in contrast with the less than 3 seconds needed by GRASP-PR. A lower bound calculation for the same problem resulted in 1 624, so the RTL-2 solution is only 1.69% higher, while the gap to the GRASP-PR solution is about 3.10%.

For another selected problem of dimension 7×7, Dr. Hahn found one solution with border length 2 290 using RTL-1 (Hahn et al., 1998), being about 1.72% better than the solution found by GRASP-PR (2 330), although it took RTL-1 some 29 hours, in contrast with the less than 7 seconds needed for the GRASP-PR run. The results obtained with exact QAP solvers give an idea of how hard the quadratic assignment problem actually is, and show that the results with GRASP-PR are a good compromise when time is limited.

Improved results for several selected problem instances from Tables 4.1 and 4.2 were also reported by Chris MacPhee using GATS, a hybrid genetic / tabu search algorithm, although these results were obtained on a number of large memory SMP machines, each having 144 processors and 576 GB of global memory. The latest results for these selected problems are available online at `http://gi.cebitec.uni-bielefeld.de/assb/chiplayout/qap`.

4.5.1 Alternatives

It is clear that, because of the large number of probes on industrial microarrays, it is not feasible to use GRASP-PR (or any other currently available QAP method) to design an entire microarray chip. However, we showed that it is certainly possible to use it on small sub-regions of a chip, which opens up the way for two alternatives.

First, the QAP approach could be used combined with a partitioning algorithm such as those discussed in Chapter 6 to the design the smaller regions that result from the partitioning. This, however, does not seem promising because, as we will see later, a partitioning is a compromise in solution quality, and level of partitioning required to achieve the dimensions supported by the QAP approach is too high.

It is interesting to extrapolate the times shown on Table 4.1 to predict the total time that would be required to design the layout of commercial microarrays, if we were to combine GRASP-PR with a partitioning algorithm. If the partitioning produced 6×6 regions, 37 636 sub-regions would be created from the 1164×1164 Affymetrix Human Genome U133 Plus 2.0 GeneChip array, one of the largest Affymetrix chips. Since each sub-region takes around 3 seconds to compute with GRASP-PR, the total time required for designing such a chip would be a little over 31 hours (ignoring the time for the partitioning itself).

If the partitioning produced 12×12 regions, 9 409 sub-regions would be created and, at 2.4 minutes each, the total time would be more than 16 days. This is probably prohibitive, although it is certainly possible to reduce the time of each GRASP-PR execution by running it on faster machines or run them in parallel.

A better alternative is to use the QAP approach to improve an existing layout, iteratively, by relocating probes inside a defined region of the chip, in a sliding-window fashion. Each iteration of this method would produce an instance of a QAP whose size equals the number of spots inside the window.The QAP heuristics could then be used to check whether a different arrangement of the probes inside the window can reduce the conflicts. For this approach to work, however, we also need to take into account the conflicts due to the spots around the window. Otherwise, a new layout with less internal conflicts could be achieved at the expense of increasing conflicts on the borders of the window.

A simple way of preventing this problem is to solve a larger QAP instance consisting of the spots inside the window as well as those in a layer (of three spots) around it. The spots outside the window obviously must remain unchanged, and that can be done by fixing the corresponding elements of the permutation π. Note that there is no need to compute f_{ij} if spots i and j are both outside the window, nor d_{kl} if probes k and l are assigned to spots outside the window.

Chapter 5

Re-embedding Algorithms

After the placement phase, it is no longer possible to reduce conflicts if probes are synchronously embedded. With asynchronous embeddings, however, layouts can usually be further improved by *re-embedding* the probes without changing their locations on the chip, in what is sometimes called a *post-placement optimization* phase.

All re-embedding algorithms discussed in this chapter are based on the Optimum Single Probe Embedding (OSPE) introduced by Kahng et al. (2002). OSPE is a dynamic programming algorithm for computing an optimum embedding of a single probe with respect to its neighbors, whose embeddings are considered as fixed. The algorithm was originally developed for border length minimization (BLM) but here we present a more general form designed for conflict index minimization (CIM) that first appeared in (de Carvalho Jr. and Rahmann, 2006a).

5.1 Optimum Single Probe Embedding

The Optimum Single Probe Embedding algorithm, OSPE for short, can be seen as a special case of a global alignment between a probe sequence p of length ℓ and the deposition sequence N of length T, disallowing mismatches and gaps in N. We assume that p is placed at spot s, and that we know the embeddings of all probes placed at spots near s (spots that are at most three cells away from s, horizontally and vertically, in accordance with the conflict index model).

The optimal embedding of p into N is built by determining the minimum cost of embedding a prefix of p into a prefix of N: We use an $(\ell + 1) \times (T + 1)$ matrix D, where $D[i, t]$ is defined as the minimum cost of an embedding of $p[1..i]$ into $N[1..t]$ for $0 \leq i \leq \ell$, $0 \leq t \leq T$. The cost is the sum of conflicts induced by the embedding of $p[1..i]$ on its neighbors (when s is unmasked and a neighbor is masked), plus the conflicts suffered by $p[1..i]$ because of the embeddings of its neighbors (when s is masked and a neighbor is unmasked).

We can compute the value for $D[i, t]$ by looking at two previous entries in the matrix: $D[i, t-1]$ and $D[i-1, t-1]$. The reason is that $D[i, t]$ is the minimum cost of embedding $p[1..i]$ up to the t-th synthesis step of N, which can only be obtained from the previous synthesis step $(t-1)$ by either masking or unmasking spot s at step t.

If s is productive (unmasked) at step t, base N_t is appended to $p[1..i-1]$; this is only possible if $p[i] = N[t]$. In this case a cost U_t is added for the risk of damaging probes at neighboring spots s'. We know that $p[1..i-1]$ can be embedded in $N[1..t-1]$ with optimal cost $D[i-1, t-1]$. Hence, the minimum cost at step t, if s is productive, is $D[i-1, t-1] + U_t$. According to the conflict index model,

$$U_t := \sum_{\substack{s':\ \text{neighbor} \\ \text{of } s}} \mathbb{1}_{\{\varepsilon_{k(s'),t}=0\}} \cdot \omega(\varepsilon_{k(s')}, t) \cdot \gamma(s', s). \tag{5.1}$$

If s is unproductive (masked) at step t, no base is appended to $p[1..i-1]$, but a cost $M_{i,t}$ must be added for the risk of damaging p (by light directed at neighboring spots s'). Since $D[i, t-1]$ is the minimum cost of embedding $p[1..i]$ in $N[1..t-1]$, the minimum cost up to step t, if s is unmasked, is $D[i, t-1] + M_{i,t}$.

Note that $M_{i,t}$ depends on the number of bases probe p already contains (that is, on i): Each unmasked neighbor s' generates a conflict on p with cost

$$\gamma(s, s') \cdot c \cdot \exp(\theta \cdot (1 + \min\{i, \ell - i\})),$$

in accordance with (2.6)–(2.8). Thus,

$$M_{i,t} := c \cdot \exp(\theta \cdot (1 + \min\{i, \ell - i\})) \cdot \sum_{\substack{s':\ \text{neighbor} \\ \text{of } s}} \mathbb{1}_{\{\varepsilon_{k(s'),t}=1\}} \cdot \gamma(s, s'). \tag{5.2}$$

Finally, $D[i, t]$ is computed as the minimum cost of the possible actions,

$$D[i, t] := \begin{cases} \min\{\, D[i, t-1] + M_{i,t},\ D[i-1, t-1] + U_t \,\} & \text{if } p[i] = N[t], \\ D[i, t-1] + M_{i,t} & \text{if } p[i] \neq N[t]. \end{cases}$$

The first column of D is initialized as follows: $D[0, 0] = 0$ and $D[i, 0] = \infty$ for $0 < i \leq \ell$, since no probe of length $\ell > 0$ can be embedded into an empty deposition sequence. The first row is initialized by setting $D[0, t] = D[0, t-1] + M_{0,t}$ for $0 < t \leq T$.

If we assume that costs U_t and $M_{i,t}$ can be computed in constant time, the time complexity of the OSPE algorithm is $O(\ell T)$ since there are $O(\ell T)$ entries in D to compute. The algorithm can be rather time-consuming in the general form presented here, since we have to look at the embeddings of up to 48 neighbors around s. Naturally, it runs

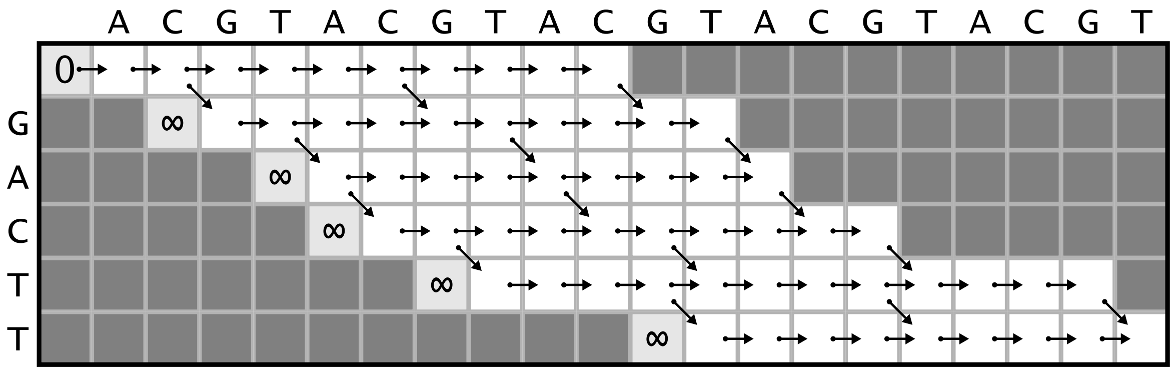


Figure 5.1: OSPE's dynamic programming matrix for computing an optimal embedding of a probe p = GACTT in a deposition sequence $N = \{\text{ACGT}\}^5$. Dark shaded cells are not computed. Arrows show all paths in the matrix leading to a valid embedding of p in N.

much faster for border length minimization, since there are only four neighbors to look at, and there are neither position-dependent (ω) nor distance-dependent (γ) weights to compute. In practice, a simple optimization can significantly reduce running time: in each row, only the columns between the left-most and right-most embeddings of p in N need to be computed (see Figure 5.1).

Once D is computed, the minimum cost is $D[\ell, T]$, and an optimal embedding of p in N can be constructed by tracing a path from $D[\ell, T]$ back to $D[0, 0]$, similarly to the procedure used to build an optimal global alignment. This takes $O(T)$ time.

The OSPE algorithm is the basic operation of several post-placement optimization algorithms: Chessboard, Greedy and Batched Greedy, and Sequential, as well as our new Priority re-embedding algorithm. The main difference between these algorithms lies in the order in which the probes are re-embedded.

Since OSPE never increases the amount of conflicts in the region around the re-embedded probe, optimization algorithms can execute several re-embedding operations without risk of worsening the current layout. Moreover, each probe may be re-embedded several times since new improvements may be possible once its neighbors have been changed. In fact, all algorithms presented here work in repeating cycles of optimization until no more improvements are possible (when a local optimal solution is found), until improvements drop below a given threshold W, or until a given number of cycles (or *passes*) have been executed.

5.2 Chessboard

The Chessboard re-embedding algorithm (Kahng et al., 2002) was initially designed for border length minimization and it takes advantage of the fact that, in this model, a

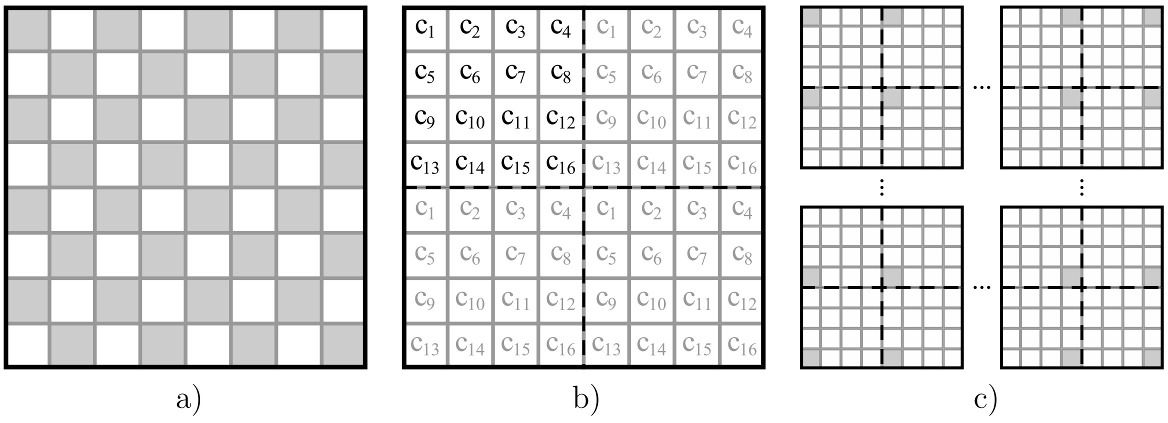

c1 c2 c3 c4 c1 c2 c3 c4
c5 c6 c7 c8 c5 c6 c7 c8
c9 c10 c11 c12 c9 c10 c11 c12
c13 c14 c15 c16 c13 c14 c15 c16
c1 c2 c3 c4 c1 c2 c3 c4
c5 c6 c7 c8 c5 c6 c7 c8
c9 c10 c11 c12 c9 c10 c11 c12
c13 c14 c15 c16 c13 c14 c15 c16
a)
b)
c)

Figure 5.2: a) The chessboard-like bi-coloring of a chip used by the Chessboard re-embedding algorithm for border length minimization; b) a possible coloring of the chip for conflict index minimization using 16 colors ($c_1 \dots c_{16}$), resulting in sets of independent spots; c) four of the 16 sets of independent spots (shaded) that can be re-embedded in the same iteration.

chip can be bi-colored like a chessboard, in such a way that the embeddings of probes located on white spots are independent of those placed on black spots (Figure 5.2a).

Chessboard uses this coloring to alternate the optimal re-embedding of probes located on black and white spots with respect to their neighbors: Each pass of Chessboard consists of re-embedding all probes of black spots and then all probe of white spots.

The chessboard coloring guarantees that probes re-embedded in the same step are independent with respect to the border length model, i.e. they can be re-embedded without affecting the border conflicts of other spots with the same color. For conflict index minimization, the same principle can be applied by using $4 \times 4 = 16$ colors instead of 2 as illustrated in Figure 5.2 (to the best of our knowledge this has not yet been implemented).

5.3 Greedy and Batched Greedy

As its name implies, the Greedy re-embedding algorithm (Kahng et al., 2002) utilizes a greedy strategy for choosing the order in which probes are re-embedded. At each iteration, Greedy examines every spot of the chip and computes the maximum reduction of border conflicts achievable by optimally re-embedding its probe. It then selects a spot with the highest gain (reduction of conflicts) and re-embeds its probe optimally, updating the gains of adjacent spots.

A faster version of this algorithm, called Batched Greedy (Kahng et al., 2002), preselects several independent spots for re-embedding and thus sacrifices its greedy nature in favour of running time by postponing the update of gains.

Like Chessboard, Greedy and Batched Greedy were initially developed for border length minmization, and they can also be extended for conflict index minimization. The main difference is that, once a probe is re-embedded, more neighbors need to be updated. For Batched Greedy, the selection of independent spots need to take into account the minimum distance of four cells (horizonally and vertically) between spots, in accordance with the conflict index model (Section 2.3). Hence fewer spots may be re-embedded in the same iteration.

5.4 Sequential re-embedding

The Sequential algorithm (Kahng et al., 2003b) employs a much simpler and, surprisingly, more efficient strategy. The algorithm just proceeds spot by spot, from top to bottom, left to right, re-embedding each probe optimally in regard to its neighbors. Once the end of the array is reached, Sequential restarts at the top left corner of the array for the next iteration.

The algorithm is not only simple but also fast since there is no need to compute achievable gains for each spot. Nonetheless, Sequential achieved the greatest reduction of border conflicts in the experiments of Kahng et al. (2003b). The authors argue that the main shortcoming of Chessboard and Greedy is that they always re-embed an independent set of spots at a time, and dropping this requirement should allow faster propagation of the effects of new embeddings and hence convergence to a better local optimum.

Tables 5.1 and 5.2 show the results of using Sequential to re-embed the probes of chips produced by the Greedy placement algorithm (Section 3.6). The chips initially contained random probes of length 25, uniformly generated, and left-most embedded in the standard Affymetrix deposition sequence. The threshold W was set to 0.2% (Sequential stopped as soon as the total reduction of conflicts in one pass droped below 0.2%). In all cases, the threshold was reached after two passes.

The reduction of conflicts achieved by Sequential were small (at most 0.579% with border length and 0.829% for CIM), which shows that there is little room for improvements once the placement is fixed. In fact, the more time is spent during placement (greater Q), the less reduction of conflicts is achieved by re-embedding. For instance, on a 300×300 chip, the reduction in average conflict index dropped by 0.12 percentage point (from 0.829% to 0.709%) when the number of candidades per spot considered by Greedy during placement was increased from 5K to 20K.

Table 5.1: Normalized border length (NBL) before and after an optimization phase with the Sequential re-embedding algorithm. Placement was produced by the Greedy placement algorithm (Section 3.6) with border length minimization, 0-threading, and number Q of candidates per spot set to 5K and 20K. The average number of passes executed by Sequential before the threshold $W = 0.2\%$ was reached is shown. The reduction of conflicts is also shown in percentage. Running times are reported in seconds and all results are averages over a set of five chips. The time spent by Sequential is also shown as a percentage of the total time (placement plus re-embedding).

	Greedy placement			Sequential re-embedding				
Dim.	Q	NBL	Time	NBL	Reduct.	Passes	Time	%Total time
300×300	5K	18.3182	98.5	18.2121	0.579%	2.0	4.8	4.617%
	20K	18.0576	577.9	17.9726	0.471%	2.0	4.8	0.830%
500×500	5K	17.5830	345.7	17.4851	0.557%	2.0	12.7	3.538%
	20K	17.3554	1 999.8	17.2779	0.446%	2.0	12.6	0.625%
800×800	5K	16.9124	916.8	16.8201	0.546%	2.0	32.6	3.437%
	20K	16.6980	5 749.7	16.6258	0.432%	2.0	32.4	0.560%

Table 5.2: Average conflict index (ACI) before and after an optimization phase with the Sequential re-embedding algorithm with $W = 0.2\%$. Placement was produced by the Greedy placement algorithm with conflict index minimization, 0-threading, and Q set to 5K and 20K.

	Greedy placement			Sequential re-embedding				
Dim.	Q	ACI	Time	ACI	Reduct.	Passes	Time	%Total time
300×300	5K	440.5166	322.4	436.8630	0.829%	2.0	188.9	36.944%
	20K	415.5003	1 818.6	412.5536	0.709%	2.0	189.9	9.457%
500×500	5K	432.3023	952.5	428.7410	0.824%	2.0	527.3	35.632%
	20K	401.4609	4 027.2	398.6096	0.710%	2.0	528.3	11.597%
800×800	5K	426.0757	2 512.1	422.6277	0.809%	2.0	1 357.9	35.087%
	20K	392.1786	11 182.8	389.3929	0.710%	2.0	1 352.5	10.790%

Although the reductions of conflicts were relatively small, Sequential required approximately half a minute to re-embedded (two times) all probes of a 800×800 chip in the BLM case, which represented about 3.44% of the aggregate time (placement and re-embedding) when $Q = 5\text{K}$ and only 0.56% when $Q = 20\text{K}$.

In some cases, Sequential even provided comparable reduction of border conflicts, in less time, than increasing Q for Greedy. For instance, on a 800×800 chip, Greedy placement with $Q = 20\text{K}$ and two passes of Sequential re-embedding produced, in approximately half of the time, a layout with only 0.14% more border conflicts than Greedy with $Q = 40\text{K}$ and no re-embedding (16.6258 NBL in 96.4 minutes versus 16.6026 in 189.0 minutes, respectively; data not shown). In other words, running Sequential is sometimes more efficient than spending more time during placement.

Figure 5.3 shows the normalized border length per masking step of a selected 500×500 chip before and after a re-embedding phase with Sequential for BLM. It is clear that the reduction of conflicts is achieved mainly between steps 45 and 65, at the expense of a small increase in conflicts in the final synthesis steps. This is a result of fixing the placement with left-most embedded probes, which leaves no room for improvements in the first masks.

In terms of CIM, the reductions were slightly higher but Sequential was over 40 times slower than in the BLM case, taking up to 36.9% of the aggregate time. This, coupled with the fact that Greedy gives significant reductions of conflicts with increasing Q even beyond 40K, makes it difficult to justify the time spent with re-embedding, unless when Q is approaching its limit (number of probes on the chip) and one is looking for the best layout possible.

Figure 5.4 shows the normalized border length per masking step of the same 500×500 chip of Figure 5.3 before and after a re-embedding phase with Sequential for CIM (placement was produced by Greedy also for CIM). Again, reduction of conflicts is restricted to the second half of synthesis steps because of the left-most embeddings, although with relatively better improvements when compared with the border length case.

Our results gives further indication that Sequential has approximately linear time complexity (if we consider that each OSPE operation can be done in constant time). Sequential performed around 19 400 re-embeddings per second in the BLM case and around 475 re-embeddings per second in the CIM case, on average.

5.5 Priority re-embedding

In this section we describe a new re-embedding algorithm, called Priority re-embedding (PR), which uses a priority queue to control the order in which probes are re-embeded.

The algorithm starts by scanning the chip for probes which have a unique embedding in the deposition sequence. These are called *pivots* and they are used as starting locations from where the re-embeddings propagate to other spots of the chip: Once a pivot is found, all of its four adjacent spots on the chip are added to the priority queue. We assume that the chip has at least one pivot, otherwise the deposition sequence could be shortened. If this is not the case, however, we can also use probes with the minimum number of embeddings among all probes as pivots.

If the probes are initially left-most embedded, every embedding with at least one productive step in the last synthesis cycle corresponds to a probe with a unique embedding. If probes are not left-most embedded, we can compute the number of embeddings $E(p, N)$ of a probe p in the deposition sequence N in $O(\ell \cdot T)$ time with dynamic programming, where ℓ is the length of the probe and T is the length of N. In practice, it is possible to compute $E(p, N)$ for a million probes in a few seconds.

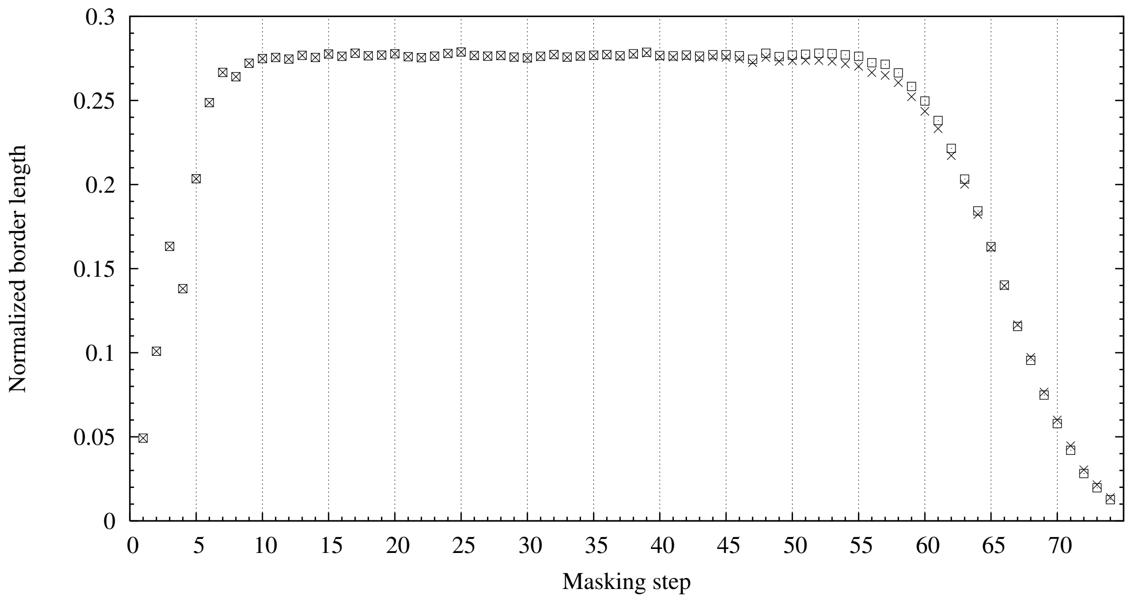


Figure 5.3: Normalized border length per masking step of a 500×500 chip before (⊡) and after (×) a re-embedding phase with Sequential for border length minimization. Layout was produced by the Greedy placement algorithm for border length minimization with 0-threading and $Q = 20$K.

The priority queue is used to retrieve the next spot s whose probe p should be re-embedded, according to the defined priority. Once a probe p is retrieved, it is optimally re-embedded in regard to its neighbors, and all four spots adjacent to s are added to the queue (if they have not been added previously).

We have implemented two different priorities: one based on the number of embeddings of each probe, and one based on the number re-embedded neighbors.

Priority I: Re-embed probes with fewer embeddings first.

The argument behind this priority is based on the observation that probes with more possible embeddings have a greater degree of freedom and can more easily “adapt” to their neighbors. Probes with a restricted number of embeddings, on the other hand, have fewer choices and should be re-embedded first.

In this priority, we examine each spot s with a probe $p_{k(s)}$ and compute $E(p_{k(s)}, N)$, the number of embeddings of $p_{k(s)}$ in N. A weight $w(s) := E(p_{k(s)}, N)$ is assigned for each spot s in the queue, and the spot with the highest weight in each iteration is retrieved.

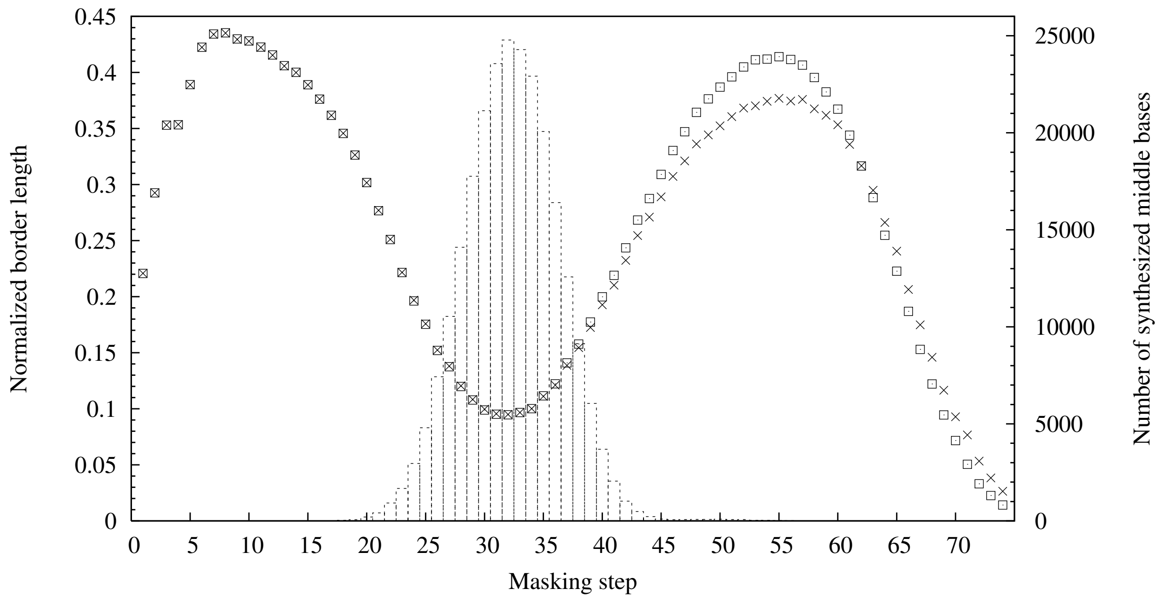


Figure 5.4: Normalized border length per masking step of a 500×500 chip before (⊡) and after (×) a re-embedding phase with Sequential for conflict index minimization. Layout was produced by the Greedy placement algorithm for conflict index minimization with 0-threading and $Q = 20\text{K}$. The number of middle bases synthesized at each step is shown in boxes (right y-axis)

Priority II: Re-embed probes with greater number of re-embedded neighbors first.

This priority tries to mimic the *seeded crystal growth* used by the Epitaxial placement algorithm (Section 3.3), giving preference to probes with a greater number of re-embedded neighbors. The argument behind this priority is that probes should not be re-embedded until a sufficient number of its neighbors have found their final embeddings.

In this priority, we also assign a weight $w(s)$ for each spot s in the queue, and the spot with the highest weight is retrieved. In case of border length minimization, $w(s)$ is set to the the number of immediate neighbors of s that have already been re-embedded in the current iteration.

In case of conflict index minimization, the algorithm looks at all 48 neighbors in the 7×7 region centered around s, and assigns a weight taking into account the distance-dependent function γ (Equation 2.5):

$$w(s) := \sum_{\substack{s':\ \text{neighbor}\\ \text{of } s}} \mathbb{1}_{\{s' \text{ has been re-embedded}\}} \cdot \gamma(s, s'),$$

where s' ranges over all neighboring spots that are at most three cells away (horizontally and vertically) from s, in accordance with the conflict index model (Section 2.3).

With Priority II, once a probe is re-embedded, it is necessary to update the weights of its neighbors that have been previously added to the queue (up to 4 with border length minimization, and 48 with conflict index minimization).

5.5.1 Results

Tables 5.3 and 5.4 show the results of using Priority re-embedding on the same set of arrays used for Sequential (Tables 5.1 and 5.2). In terms of BLM, both priorites resulted in negligible improvements when compared to Sequential (with Priority I giving the best results). The greatest difference was only 0.0032% (from 18.2121 with Sequential to 18.2115 with Priority I on 300×300 chips and Greedy placement with $Q = 5\text{K}$). Moreover, Priority I was between 8.8% and 12.7% slower than Sequential, whereas Priority II was between 2 to 5 times slower than Sequential.

Priority II is slower than Priority I because after it re-embeds a spot s, it needs to update the weights of all neighbors of s that have been previously added to the queue. With Priority I, the nuber of embeddings of each probe does not change, so they are computed only once, before the first iteration.

In terms of CIM, Priority I produced the worse layouts, whereas Priority II once again achieved negligible improvements when compared to Sequential — at most 0.0029% (from 412.5536 to 412.5418 on 300×300 chips and Greedy placement with $Q = 20\text{K}$). The difference in running times between Sequential and Priority dropped in comparison with the same difference in the BLM case. This is because OSPE is significantly slower with CIM, so the extra time spent re-embedding probes reduces the impact of the extra work with the priority queue. For this reason, Priority I was always within 0.1% of the time required by Sequential, whereas Priority II was at most 11.37% slower.

5.6 Summary

In this chapter, we have presented an extension of the Optimum Single Probe Embedding algorithm (OSPE) of Kahng et al. (2002) that is general enough to work with border length as well as conflict index minimization. We have also surveyed re-embeddings algorithms based on OSPE and presented experimental results with Sequential, the best known algorithm to date.

Table 5.3: Normalized border length (NBL) before and after an optimization phase with various re-embedding algorithms. Placement was produced by the Greedy placement algorithm with border length minimization, 0-threading, and number Q of candidates per spot set to 5K and 20K. In all cases, each re-embedding algorithm executed two passes before the threshold $W = 0.2\%$ was reached. Best results are highlighted in bold.

	Greedy placement		Sequential		Priority I		Priority II	
Dim.	Q	NBL	NBL	Time	NBL	Time	NBL	Time
300×300	5K	18.3182	18.2121	4.8	**18.2115**	5.4	18.2118	22.0
	20K	18.0576	17.9726	4.8	**17.9721**	5.4	17.9723	14.5
500×500	5K	17.5830	17.4851	12.7	**17.4848**	13.9	17.4849	76.7
	20K	17.3554	17.2779	12.6	**17.2776**	13.7	17.2777	63.9
800×800	5K	16.9124	16.8201	32.6	**16.8198**	36.1	16.8199	187.0
	20K	16.6980	16.6258	32.4	**16.6256**	35.3	16.6257	200.0

Table 5.4: Average conflict index (ACI) before and after an optimization phase with various re-embedding algorithms. Placement was produced by the Greedy placement algorithm with conflict index minimization, 0-threading, and number Q of candidates per spot set to 5K and 20K. In all cases, each re-embedding algorithm executed two passes before the threshold $W = 0.2\%$ was reached. Best results are highlighted in bold.

	Greedy placement		Sequential		Priority I		Priority II	
Dim.	Q	NBL	NBL	Time	NBL	Time	NBL	Time
300^2	5K	440.5166	436.8630	188.9	436.8881	190.7	**436.8626**	209.0
	20K	415.5003	412.5536	189.9	412.5613	190.0	**412.5418**	205.1
500^2	5K	432.3023	428.7410	527.3	428.7640	527.2	**428.7375**	581.6
	20K	401.4609	398.6096	528.3	398.6261	530.0	**398.6065**	569.5
800^2	5K	426.0757	422.6277	1357.9	422.6478	1357.9	**422.6223**	1512.2
	20K	392.1786	389.3929	1352.5	389.4075	1355.3	**389.3903**	1488.9

In our results, it is evident that there is little room for improvements by re-embedding probes once a placement is fixed. Nonetheless, we have also introduced a new re-embedding algorithm that attempts to obtain better results by changing the order of re-embeddings based on priorities. We have experimented with two priorities: probes with fewer embeddings first (Priority I) and probes with more re-embedded neighbors (Priority II).

Our results show that our algorithm can achive negligible improvements when compared to Sequential, with Priority I being the best for BLM and Priority II the best for CIM. However, because of the extra time required by Priority, Sequential offers a better trade-off between solution quality and running time, and it should still be the algorithm of choice unless when time is not constrained. The results with our new algorithm also give further indication that the improvements achievable in the re-embedding phase are rather small.

Chapter 6

Partitioning Algorithms

We mentioned earlier that the Microarray Layout Problem is usually approached in two phases: placement and re-embedding. The placement, however, can be preceded by a *partitioning* phase that breaks the problem into smaller sub-problems that are easier to manage. This is especially helpful for placement algorithms with non-linear time or space complexities that are otherwise unable to handle very large chips.

A partitioning algorithm divides the set of probes $\mathcal{P}$ into smaller subsets, and assigns them to defined regions of the chip. Each region can then be treated as an independent chip (and processed by a placement algorithm) or be recursively partitioned. Linear-time placement algorithms may also benefit from a partitioning since probes with similar embeddings are typically assigned to the same region — Greedy and Row-Epitaxial (Chapter 3), for instance, are more likely to find good candidates for filling the spots.

We describe four partitioning algorithms: 1-Dimensional Partitioning (1-DP), 2-Dimensional Partitioning (2-DP), Centroid-based Quadrisection (CQ), and Pivot Partitioning (PP). Like placement algorithms, they assume that an initial (left-most, right-most, synchronous or otherwise pre-computed) embedding of the probes is given. Pivot Partitioning is the only algorithm that modifies these embeddings. As we shall see, 1-DP and 2-DP generate a few masks with extremely few conflicts, but leave the remaining masks with high levels of conflicts that are difficult to handle. CQ and PP offer a more uniform optimization over all masks. Earlier results indicate that PP produces better layouts than CQ on large chips (de Carvalho Jr. and Rahmann, 2006a).

Partitioning is clearly a compromise in solution quality since it restricts the space of solutions and may lead to conflicts at partition borders, although it can improve solution quality when the placement algorithm cannot handle large regions well. Hence, it is not advisable to perform too many levels of partitioning because smaller sub-regions mean less freedom for optimization during placement. The right balance depends on the chip dimensions as well as on the placement and partitioning algorithms.

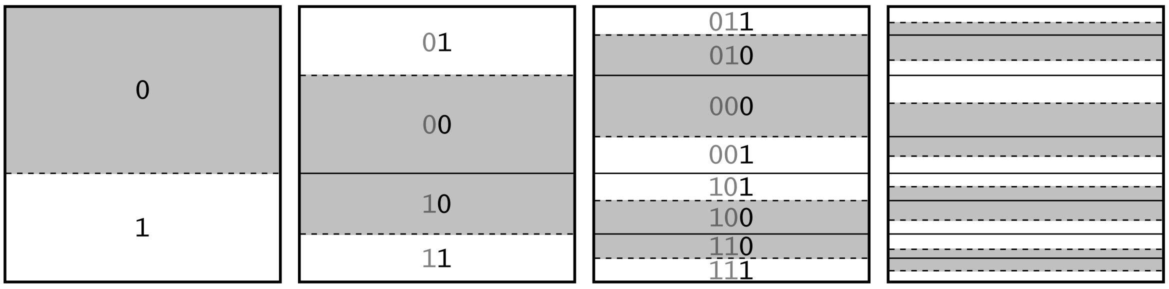


Figure 6.1: First four levels of 1-Dimensional Partitioning. Dashed lines show the divisions performed in each step; solid lines indicate regions delimited in previous steps (there are no border conflicts between spots separated by solid lines). Masked (shaded) regions are labeled "0", unmasked (white) regions are labeled "1". This labeling forms a binary Gray code (shown in the first three steps only).

6.1 1-Dimensional Partitioning

1-Dimensional Partitioning (1-DP) divides the set of probes based on the state of their embeddings at a particular synthesis step. It starts by creating two subsets of $\mathcal{P}$:

$$\mathcal{P}_0 = \{p_k \in \mathcal{P} | \varepsilon_{k,1} = 0\}, \qquad \mathcal{P}_1 = \{p_k \in \mathcal{P} | \varepsilon_{k,1} = 1\}.$$

In other words, $\mathcal{P}_0$ contains all probes whose embeddings are unproductive during the first synthesis step, whereas $\mathcal{P}_1$ contains probes with productive embeddings. The chip is then divided into two horizontal (or vertical) bands, proportionally to the number of probes in $\mathcal{P}_0$ and $\mathcal{P}_1$, so each band accommodates one subset of $\mathcal{P}$.

This procedure is recursively applied to each band, using the the next synthesis steps to further divide each subset of probes. For instance, the following subsets of $\mathcal{P}_0$ and $\mathcal{P}_1$ are created during step $t = 2$:

$$\mathcal{P}_{00} = \{p_k \in \mathcal{P}_0 | \varepsilon_{k,2} = 0\}, \qquad \mathcal{P}_{01} = \{p_k \in \mathcal{P}_0 | \varepsilon_{k,2} = 1\},$$

$$\mathcal{P}_{10} = \{p_k \in \mathcal{P}_1 | \varepsilon_{k,2} = 0\}, \qquad \mathcal{P}_{11} = \{p_k \in \mathcal{P}_1 | \varepsilon_{k,2} = 1\}.$$

The next assignments of subsets to the upper or lower band of their regions are made in such a way that regions with the same "state" — productive (unmasked) or unproductive (masked) — are joined as far as possible, resulting in masks that consist of alternating layers of masked and unmasked spots. This process is illustrated in Figure 6.1, where at each step t, a band is labeled "0" when its embeddings are unproductive, and "1" when its embeddings are productive. The resulting binary numbers from top to bottom form a binary Gray code, that is, a permutation of the binary numbers between 0 and $2^n - 1$ such that neighboring elements have exactly one differing bit, as do the first and last elements (Kreher and Stinson, 1999).

The Gray code highlights an interesting property of 1-DP. After d levels of partitionings (based on steps 1 to d), the embeddings of any two immediate neighbors differ among the first d steps in at most one step. As a result, masks $M_1 \dots M_d$ exhibit a layered structure that effectively reduces border conflicts.

Unfortunately, the Gray code is disrupted as soon as a region cannot be divided (because all probes of that region are, for instance, masked at a particular step). This will certainly happen as several binary numbers are unlikely to be substrings of embeddings (think of, for example, a long run of zeros).

Moreover, 1-DP can optimize only a limited number of masks because the sub-regions soon become too narrow to be further divided. The maximum *partitioning depth* d_{max} is primarily limited by the number of rows (or columns) on the chip. In practice, since regions are likely to be unevenly divided, d_{max} varies between regions. The algorithm can also be configured to stop partitioning a region once its height drop below a given threshold H_{max} (i.e., the maximum height of any final region will not exceed H_{max}).

1-DP is easier to implement if the partitionings always produce rectangular regions (i.e., splitting a row between two regions is not allowed). In order to force an exact division of a region, however, it might be necessary to move a few probes from one subset of probes to the other one.

For example, imagine that a chip with $|\mathcal{P}| = 900$ probes, $n_r = 30$ rows and $n_c = 30$ columns is to be partitioned based on the state of the embeddings at the first synthesis step, resulting in sub-sets $\mathcal{P}_0$ and $\mathcal{P}_1$ with, say, 638 and 262 probes, respectively. The chip must thus be divided into two sub-regions, the upper one containing $[30 \cdot 638/900] = 21$ rows and the lower one with $[30 \cdot 262/900] = 9$ rows ($[x]$ is x rounded to the nearest integer). The problem is that the upper region then contains $21 \cdot 30 = 630$ spots but it has to accommodate 638 probes, whereas the lower region contains $9 \cdot 30 = 270$ spots but only 262 probes. The solution is to (arbitrarily) move 8 probes from $\mathcal{P}_0$ to $\mathcal{P}_1$, which, results in some imperfections in the layers of the corresponding mask (a few masked spots in a region of unmasked spots and vice-versa).

6.2 2-Dimensional Partitioning

The 2-Dimensional Partitioning algorithm extends the idea of 1-DP to two dimensions, with the potential of optimizing twice as many masks. The algorithm is similar: $\mathcal{P}$ is divided into subsets based on the state of the embeddings at a particular synthesis step. The differences are that 2-DP alternates horizontal and vertical divisions of regions, and that the assignments of probes to regions obey a two-dimensional binary Gray code (Figure 6.2). In a 2-D Gray code, the binary numbers are arranged in a matrix in such a way that two neighboring numbers differ in at most one bit. As a

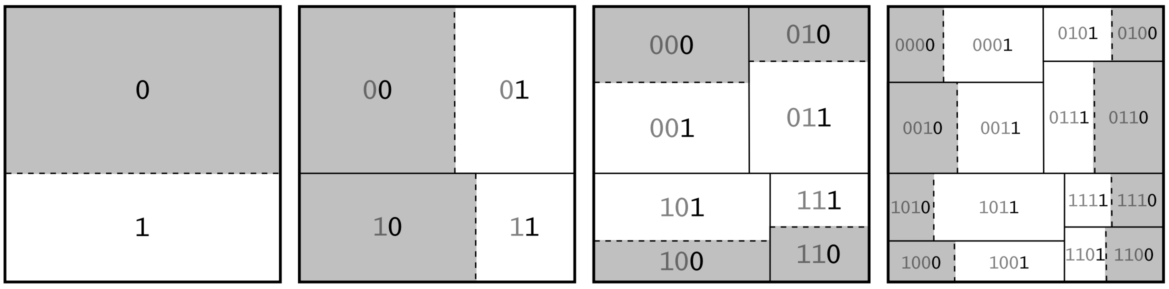


Figure 6.2: First four levels of 2-Dimensional Partitioning. Dashed lines show the divisions performed in each step; solid lines indicate regions delimited in previous steps. Masked regions are labeled with "0", unmasked regions with "1"; this labeling forms an approximation to a two-dimensional binary Gray code.

result, regions whose embeddings are at the same state (productive or unproductive) are joined as far as possible.

If regions were always equally divided, 2-DP would have the same property as 1-DP: After d levels of partitionings (based on steps 1 to d), the embeddings of any two immediate neighbors would differ among the first d steps in at most one step. However, this is not always the case since 2-DP is likely to create regions with different dimensions, forcing some regions to share a border with more than its four natural neighbors. For instance, in Figure 6.2 region "0010" borders with "0000", "1010", and "0011", but also with "0001" and "1011".

Like 1-DP, the maximum partitioning depth, d_{max}, is limited by the number of rows and columns on the chip, and it varies since regions are likely to be unevenly divided. 2-DP can also be configured to stop partitioning a region once its dimensions (height and width) drop bellow a given threshold L_{max} (the largest final region will contain at most ${L_{max}}^2$ spots).

Figure 6.3 shows the normalized border length per masking step of layouts produced by 2-DP for a $1\,000 \times 1\,000$ chip. With maximum partitioning depth ($L_{max} = 1$), 2-DP produced a layout with the best masks for the first 22 synthesis steps. However, because the chip is partitioned until all regions are 1×1 (containing a single probe), the placement algorithm has no freedom for reducing border conflicts in the remaining masks. As a result, from step 33 on, the levels of border conflicts are as high as in the random layout.

With $L_{max} = 10$, there is more room for optimization during placement since the final regions can be as large as 10×10. In this case, we used the Greedy placement algorithm (Section 3.6) with $Q = 100$ so that all probes of a region were considered for filling its spots. This resulted in a reduction of about 13.4% in normalized border length compared to the layout produced with $L_{max} = 1$ (from 21.5588 to 18.6670, data not shown), although we observed an increase of border conflicts in the first 24 masks.

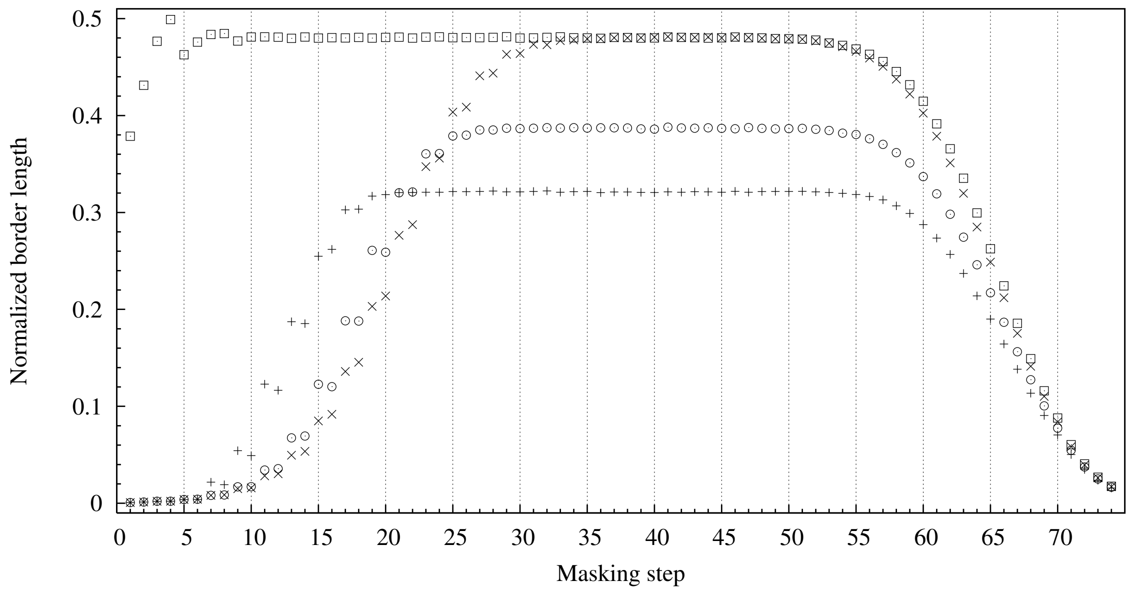


Figure 6.3: Normalized border length per masking step of several layouts for a 1 000 × 1 000 chip with random probes left-most embedded in the standard Affymetrix depostion sequence: random layout (⊡); 2-D Partitioning with $L_{max} = 1$ (×); 2-D Partitioning with $L_{max} = 10$ and Greedy placement with $Q = 100$ (⊙); 2-D Partitioning with $L_{max} = 50$ and Greedy placement with $Q = 2\,500$ (+).

Increasing L_{max} even further to 50 and using Greedy with $Q = 2\,500$ resulted in a reduction of 8.1% in normalized border length compared to $L_{max} = 10$ (from 18.6670 to 17.1629) but, again, this came at the expense of an increase of border conflicts in the first 20 masks.

Figure 6.4 compares the results obtained by 1-DP and 2-DP on the same 1 000 × 1 000 chip of Figure 6.3. We first compare both algorithms with their maximum partitioning depths ($H_{max} = 1$ for 1-DP and $L_{max} = 1$ for 2-DP). With $L_{max} = 1$, 2-DP produces 1 × 1 regions and leaves no room for optimization during placement. In constrast, 1-DP with $H_{max} = 1$ produces regions with a single row but, in this case, with 1 000 columns (and 1 000 probes), leaving a considerable degree of flexibility for the placement algorithm. To be fair, we thus compare 1-DP and 2-DP using a placement algorithm that places probes randomly inside each final region, so that the results are only due to the partitionings (and not to the placement algorithm). In our results, with maximum partitioning depths, 1-DP and 2-DP produced layouts with similar levels of border conflicts in mask $M_{33} \ldots M_{74}$, although the layout produced by 2-DP was slightly better in masks $M_{58} \ldots M_{69}$. However, while 1-DP was able to produce masks with relatively few conflicts in the first 17 steps, 2-DP achieved even greater reductions of border conflicts in the first 32 steps. The normalized border length of these layouts are 25.8543 (with 1-DP) and 21.5588 (with 2-DP).

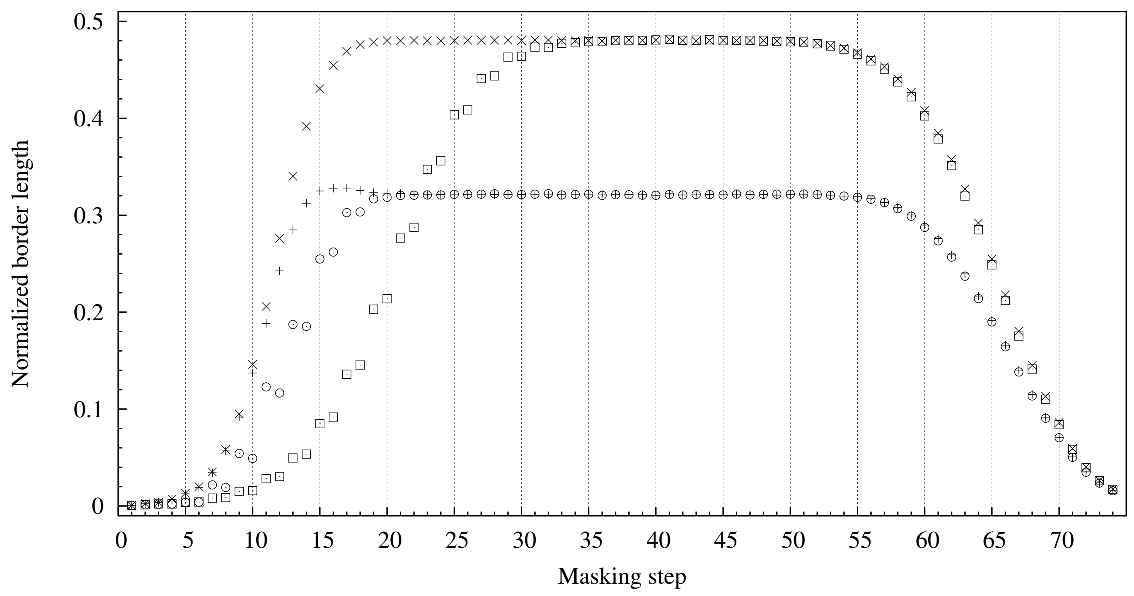


Figure 6.4: Normalized border length per masking step of layouts produced by 1-D and 2-D Partitioning for a 1 000 × 1 000 chip with random probes left-most embedded in the standard Affymetrix depostion sequence: 1-DP with $H_{max} = 1$ and random placement (×); 2-DP with $L_{max} = 1$ (⊡); 1-DP with $L_{max} = 1$ and Greedy placement with $Q = 1$K (+); 2-DP with $L_{max} = 50$ and Greedy placement with $Q = 2.5$K (⊙) .

In Figure 6.4 we also compare 1-DP with $H_{max} = 1$ and 2-DP with $L_{max} = 50$ using Greedy for the placement. With $L_{max} = 50$, 2-DP produces regions containing at most 2 500 probes. For this particular chip, 2-DP produced 1 005 regions, containing 995.02 probes on average (the largest region contained 2 209 and the smallest 210 probes), so Greedy had about the same degree of freedom provided by 1-DP with $H_{max} = 1$. We used a sufficiently large number Q of candidates per spot so that all probes of a region were considered for filling its spots. With these settings, the layouts produced by 1-DP and 2-DP have similar levels of border conflicts in masks $M_{20} \dots M_{74}$. In the first 18 synthesis steps, however, 2-DP produced better masks, especially after step 5. The normalized border length of these layouts are 18.0078 (1-DP) and 17.1629 (2-DP).

A representation of selected photolithographic masks generated by 2-DP for a 300×300 chip are shown in Figure 6.5. The resulting rectangular regions can be seen, clearly, up to mask M_{18}. In the first eight masks it is possible to see some “imperfections” (unmasked spots on masked regions or vice-versa) that result from arbitrarily moving probes between regions in order to force exact divisions. On a chip of this size, 2-DP can usually reduce conflicts up to the 25th synthesis step, although this is not noticeable in M_{25} of Figure 6.5.

So far we have described both 1-DP and 2-DP using the state of the first d synthesis steps to divide the set of probes. The result of this approach is that, while the first

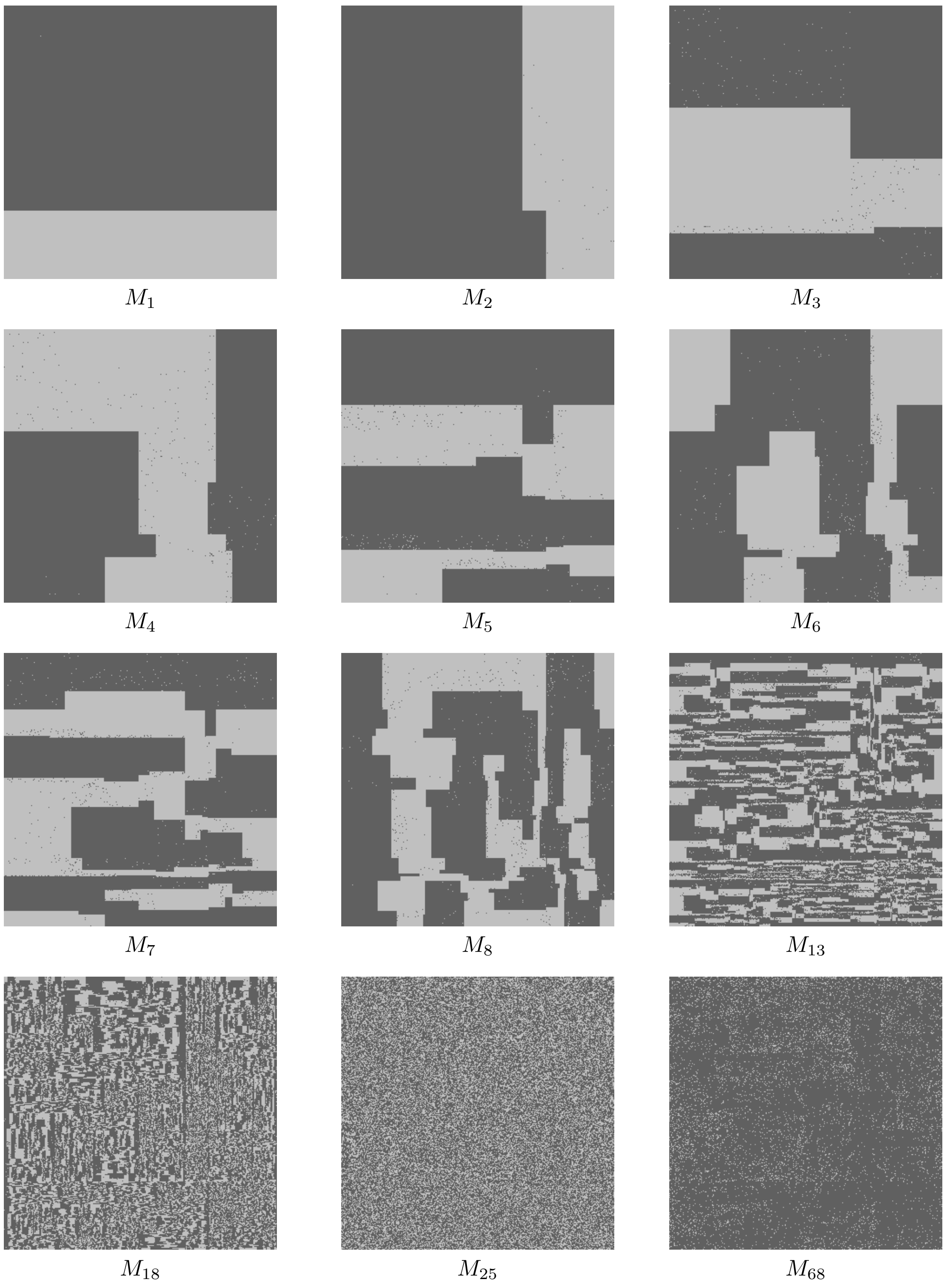


Figure 6.5: Selected masks generated by 2-Dimensional Partitioning with $L_{max} = 1$ for a random 300×300 chip with 25-mer probes leftmost embedded into the standard Affymetrix deposition sequence. Unmasked (masked) spots are represented by light (dark) dots.

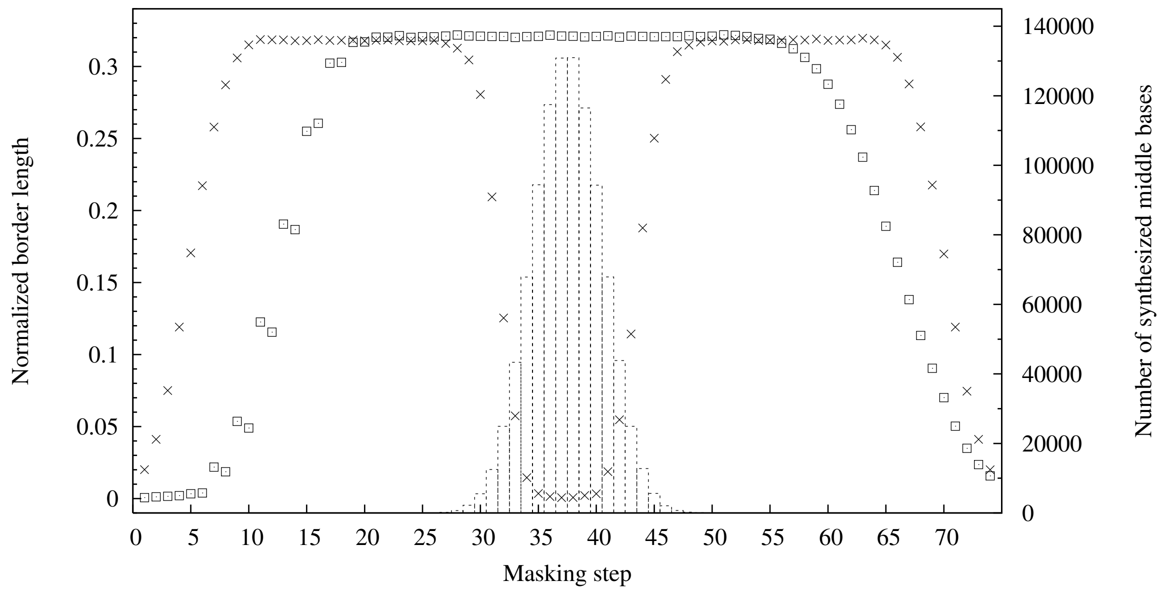


Figure 6.6: Normalized border length per masking step (on the left y-axis) of two layouts produced by 2-Dimensional Partitioning with $L_{max} = 50$ and Greedy placement with border length minimization and $Q = 2.5$K for a $1\,000 \times 1\,000$ chip with random probe sequences: left-most mask optimization with left-most embeddings (⊡); centered mask optimization with centered embeddings ($\times$). The number of middle bases synthesized at each step (with centered embeddings) is shown in boxes (right y-axis).

masks are optimized, the remaining masks are left with high levels of border conflicts; we call this a *left-most mask optimization*.

However, a defect in the middle of the probe is more harmful than in its extremities, so it is more important to optimize the central masks that are more likely to synthesize the probes' middle bases. Fortunately, it is possible to reduce conflicts in the central masks using 1-DP and 2-DP by partitioning the probe set based on the following sequence of synthesis steps, assuming that T is even and d is odd: $T/2, (T/2) \pm 1, (T/2) \pm 2, \ldots, (T/2) \pm \lfloor d/2 \rfloor$; we call this a *centered mask optimization*.

For left-most optimization, it makes sense to embed the probes in a left-most fashion in order to reduce conflicts in the last masks (which are not optimized by the partitioning). The left-most embeddings reduce the number of unmasked spots in the last steps, resulting in masks that largely consist of masked spots and consequently low levels of border conflicts. In contrast, centered mask optimization produces better results with *centered* embeddings. A centered embedding is constructed by shifting a left-most embedding to the right until the number of masked steps to the left of the first productive step is approximately equal to the number of masked steps to the right of the last productive step.

Table 6.1: Average conflict index (ACI) of layouts produced by Greedy placement and 2-D Partitioning on random 800 × 800 chips with left-most and centered embeddings. 2-DP was configured for centered mask optimization and used Greedy for the placement. In all cases, Greedy was configured for conflict index minimization and used 0-threading. Results are averages over a set of five arrays and running times are reported in minutes.

Embeddings	Algorithm	ACI	Time
Leftmost	Greedy with $Q = 20\text{K}$	392.1786	186.4
Leftmost	Greedy with $Q = 40\text{K}$	378.3110	357.0
Leftmost	Greedy with $Q = 80\text{K}$	366.8446	680.9
Centered	Greedy with $Q = 20\text{K}$	387.5974	205.1
Centered	2-DP with $L_{max} = 10$ and Greedy with $Q = 100$	345.9908	0.6
Centered	2-DP with $L_{max} = 20$ and Greedy with $Q = 400$	342.2031	1.3
Centered	2-DP with $L_{max} = 30$ and Greedy with $Q = 900$	**341.2786**	2.3
Centered	2-DP with $L_{max} = 40$ and Greedy with $Q = 1\,200$	341.6185	4.0
Centered	2-DP with $L_{max} = 50$ and Greedy with $Q = 2\,000$	341.7515	6.1
Centered	2-DP with $L_{max} = 60$ and Greedy with $Q = 3\,600$	341.8634	8.4

Figure 6.6 shows the results of using 2-D Partitioning with $L_{max} = 50$ on a $1\,000 \times 1\,000$ chip with left-most and centered mask optimization. With left-most mask optimization, we obtain a normalized border length of 17.1629 (up to approximately 0.32 per step). With centered mask optimization, the normalized border length improves by 1.03% to 16.9855 (not shown in the figure). The average conflict index, however, is reduced by as much as 34.89% (from 577.3353 to 375.9232) because of the higher weight of the middle bases in the conflict index measure.

When carefully used, 1-DP and 2-DP can improve placement by producing a few masks with very low levels of border conflicts and breaking the problem into smaller subproblems that are easier to handle. Table 6.1 shows results on 800 × 800 arrays using 2-DP with centered mask optimization and Greedy with conflict index minimization for the placement, in comparison to using Greedy alone (results with Greedy as shown on Table 3.2 and Figure 3.5). Results of Greedy with centered embeddings are also shown. In our results, the layouts produced by 2-DP are even better than the ones produced by Greedy with $Q = 80\text{K}$. This is a consequence of the importance of the middle bases in the conflict index measure. Moreover, while Greedy required about 680.9 minutes with $Q = 80\text{K}$, the combination of 2-DP and Greedy required at most 8.4 minutes because the partitioning restricts the number of candidates Greedy can look at for each spot.

Increasing L_{max} provides more room for optimization during placement but worsens the central masks, while reducing L_{max} improves the central masks at the expense of an increase of conflicts in the remaining masks (in this case, reducing L_{max} also improves running time as Greedy has fewer candidates available for each spot). The best trade-off depends on several aspects of the problem such as chip dimension, probe embeddings, type of optimization (border length or conflict index), and placement algorithm. For this case, the best results were achieved with $L_{max} = 30$.

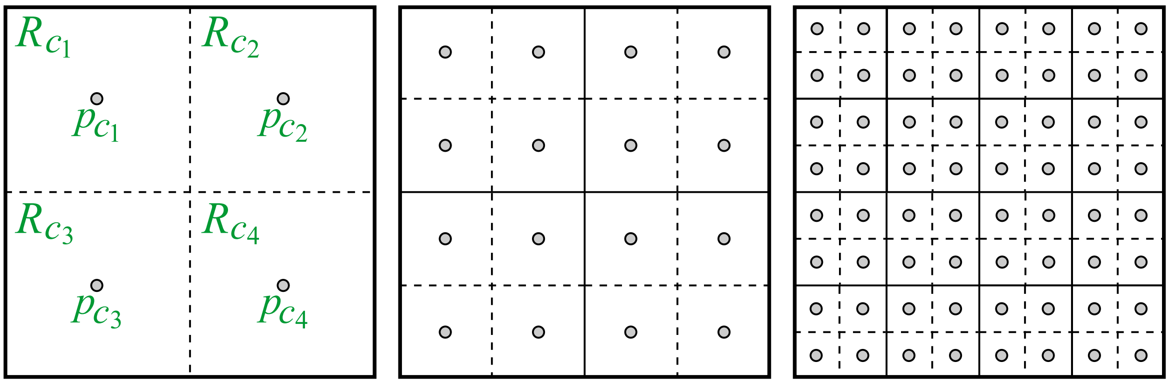


Figure 6.7: First three levels of Centroid-based Quadrisection. Dashed lines show the divisions performed in each step; solid lines indicate regions delimited in previous steps. The centroids of each partition $R_{c_1} \dots R_{c_4}$ are represented by small circles (labeled with $p_{c_1} \dots p_{c_4}$ in the first step).

6.3 Centroid-based Quadrisection

Centroid-based Quadrisection (Kahng et al., 2003b), CQ for short, employs a different criterion for dividing the probe set and a different approach for partitioning. At each iteration, a region R is quadrisectioned into R_{c_1}, R_{c_2}, R_{c_3}, and R_{c_4}. Each sub-region R_{c_i} is associated with a selected probe $p_{c_i} \in \mathcal{P}$, called *centroid*, that is used to guide the assignment of the remaining probes to the sub-regions.

A centroid is a representative of its region: It should symbolize the "average embedding" in that region. The remaining probes $p_k \in \mathcal{P} \setminus \{p_{c_1}, p_{c_2}, p_{c_3}, p_{c_4}\}$ are compared to each centroid and assigned to the sub-region R_{c_i} whose centroid's embedding ε_{c_i} has minimum $H(k, c_i)$, where $H(k, k')$ is the Hamming distance between the embeddings ε_k of p_k and $\varepsilon_{k'}$ of $p_{k'}$ (see Section 2.2).

The authors argue that, in order to improve the "clustering" of similar probes, the four centroids should be as different from each other as possible. The following heuristic is proposed: First, a probe index c_1 is randomly selected from $\{1, \dots, |\mathcal{P}|\}$. Then, a probe index $c_2 \neq c_1$ maximizing $H(c_2, c_1)$ is selected. Similarly, c_3 maximizing $H(c_3, c_1) + H(c_3, c_2)$ and c_4 maximizing $H(c_4, c_1) + H(c_4, c_2) + H(c_4, c_3)$ are selected. The assignment of centroids to the quadrisections of the chip is arbitrary.

Since the partitioning must always produce four regions of the same size, sometimes it is necessary to make non-optimal assignment of probes to regions. In order to recover from a possibly bad choice of centroids, a "multi-start heuristic" is used, running the centroid selection procedure several times with different "seeds" for c_1 and keeping the centroids that lead to the best partitioning. For measuring partitioning quality, the algorithm uses the sum of Hamming distances between the embeddings of the probes and the embedding of the centroid (the partitioning that results in the least sum is selected).

The maximum partitioning depth d_{max} of CQ is $\sqrt{n_r}$, assuming that n_r is square and that $n_c = n_r$ (n_r and n_c are the number of rows and columns on the chip, respectively). In practice, the partitioning continues until a pre-defined depth D has been reached.

CQ was developed for border length minimization (BLM), but it can be adapted for conflict index minimization (CIM) by using the *conflict index distance* $C(k, k')$ between the embeddings ε_k and $\varepsilon_{k'}$ (as defined in Section 2.3) instead of the Hamming distance $H(k, k')$ for selecting the centroids as well as for deciding which partition a probe should be assigned to.

As mentined in Section 3.6, placement algorithms such as Row-Epitaxial and Greedy have the drawback of treating the last $Q - 1$ filled spots unfairly since fewer than Q probe candidates are available to fill them. This issue is aggravated by a partitioning because in each final partition $Q - 1$ spots have fewer than Q probe candidates. In order to attenuate this problem, a *borrowing heuristic* was implemented in CQ to allow the placement algorithm (Row-Epitaxial, in the original implementation) to look at Q probes "in the current and the next region". Although the authors did not specify the exact meaning of "next region", it can be, for instance, the next region to be processed by the placement algorithm. Borrowing probes from a region R_{c_i} to fill spots of R_{c_j} obviously requires using the unplaced probes of R_{c_j} to fill spots of R_{c_i}.

6.4 Pivot Partitioning

Pivot Partitioning (de Carvalho Jr. and Rahmann, 2006a), PP for short, is to a certain extent similar to CQ: Sub-regions are recursively associated with special probes, here called *pivots* instead of centroids, that are used to guide the assignment of the other probes to the sub-regions. The main differences between PP and CQ are as follows.

Instead of quadrisectioning the chip, PP creates sub-regions by alternating horizontal and vertical divisions (like 2-D Partitioning). At each iteration, a region R is partitioned into sub-regions R_{c_1} and R_{c_2} associated with pivots q_{c_1} and q_{c_2}, respectively. The advantage of alternating horizontal and vertical divisions over the quadrisectioning approach of CQ is that regions are not required to have the same size. Instead, regions are divided proportionally to the size of each subset of probes, which reduces the need for making non-optimal assignments, although it may still be necessary to move some probes from one sub-region to the other in order to obtain rectangular regions. Moreover, for each partitioning, only two pivots need to be selected.

Another distinction is motivated by the same observation that inspired the development of the Priority re-embedding algorithm (Section 5.5), i.e., that different probes

Algorithm 1 PivotPartitioning

```
Input:  rectangular region R consisting of all rows and columns of the chip,
        set of probes $\mathcal{P} = \{p_1, p_2, \ldots p_n\}$,
        deposition sequence $N$,
        and maximum partitioning depth $D$
Output: set of assignments $\mathcal{A} = \{a_1, a_2, \ldots a_{2^D}\}$
        where $a_i = (\mathcal{P}_i, R_i)$, $\mathcal{P}_i \subset \mathcal{P}$, and $R_i$ is a sub-region of the chip

1. (Select pivot candidates.) Select probes $p \in \mathcal{P}$ with minimum number of embeddings $E(p)$ as
   pivot candidates:
    a) Let $\mathcal{Q} = \{p \in \mathcal{P} \mid E(p, N) \text{ is minimal}\}$
    b) Set $\mathcal{P} \leftarrow \mathcal{P} \setminus \mathcal{Q}$
2. (Call RecursivePartitioning.) Call recursive procedure with initial partitioning depth 1 and
   return:
    a) Return RecursivePartitioning $(1, D, R, \mathcal{Q}, \mathcal{P})$
```

have different numbers of embeddings, ranging from a single one to several millions on a typical Affymetrix GeneChip array. Probes with more embeddings can more easily adapt to the other probes, that is, they are more likely to have an embedding with fewer conflicts to fill a particular spot than a probe that has only a limited number of embeddings. PP uses probes with a single embedding (or few embeddings) as pivots, and chooses the other probes' embeddings and region assignments accordingly. Indeed, the most important feature of PP is the simultaneous embedding and assignment of probes to sub-regions.

The first part of the algorithm consists of selecting a sub-set of probes that will be used as pivots (Algorithm 1). First, it examines each probe $p \in \mathcal{P}$ and computes $E(p, N)$, the number of embeddings of p in the deposition sequence N; this can be done in $O(\ell \cdot T)$ time with dynamic programming, where ℓ is the length of the probe and T is the length of the deposition sequence. The set of pivot candidates $\mathcal{Q}$ then consists of all probes p with $E(p, N) = 1$. In practice, this usually results in a sufficient number of pivots. For instance, around 6% of the probes in a randomly generated chip have a single embedding. If this is not the case, we can set a threshold e for the maximum number of embeddings of a pivot in such a way that the number of probes p with $E(p, N) \leq e$ is at least 2^D.

Using probes with fewer embeddings as pivots has two advantages. First, less time is spent choosing the pivots in each iteration since fewer candidates need to be examined. Second, probes with fewer embeddings are usually better "representatives" to drive the partitioning. The problem is that some embeddings may have their productive steps concentrated in one part of the deposition sequence. For instance, some Affymetrix probes, when left-most embedded, are synthesized in the first 37 masking steps, thus using only half of the total 74 steps. Such probes are not good choices for pivots. In our experience, probes with fewer embeddings are better pivots because they cover

Algorithm 2 RecursivePartitioning with conflict index minimization

```
Input:  current partitioning depth $d$,
        maximum partitioning depth $D$,
        rectangular region $R$ of the chip,
        set of pivot candidates $\mathcal{Q}$,
        and set of probes $\mathcal{P}$,
Output: set of assignments $\mathcal{A} = \{a_1, a_2, \dots a_{2^{(D-d)}}\}$
        where $a_i = (\mathcal{P}_i \cup \mathcal{Q}_i, R_i)$, $\mathcal{P}_i \subset \mathcal{P}$, $\mathcal{Q}_i \subset \mathcal{Q}$, and $R_i$ is a sub-region of $R$

1. (Stopping condition.) When $d = D$:
    a) Re-embed each $p \in \mathcal{P}$ optimally with respect to all $q \in \mathcal{Q}$
    b) Return $(\mathcal{P} \cup \mathcal{Q}, R)$
2. (Choose pivot pair.) Select $q_{c_1}, q_{c_2} \in \mathcal{Q}$ such that $C(c_1, c_2)$ is maximal
3. (Partition set of pivot candidates.) Assign each pivot candidate $q_k \in \mathcal{Q}$ to sub-set $\mathcal{Q}_{c_j}$ associated with pivot $q_{c_j}$ such that $C(k, c_j)$ is minimal; in case of ties, make assignments heuristically in an attempt to achieve balanced partitionings:
    a) $\mathcal{Q}_{c_1} = \{q_k \in \mathcal{Q} \mid C(k, c_1) < C(k, c_2)\}$
    b) $\mathcal{Q}_{c_2} = \{q_k \in \mathcal{Q} \mid C(k, c_1) > C(k, c_2)\}$
4. (Partition probe set.) Assign each probe $p_k \in \mathcal{P}$ to sub-set $\mathcal{Q}_{c_j}$ such that $M_C(k, c_j)$ is minimal; in case of ties, make assignments heuristically in an attempt to achieve balanced partitionings:
    a) $\mathcal{P}_{c_1} = \{p_k \in \mathcal{P} \mid M_C(k, c_1) < M_C(k, c_2)\}$
    b) $\mathcal{P}_{c_2} = \{p_k \in \mathcal{P} \mid M_C(k, c_1) > M_C(k, c_2)\}$
5. (Partition chip region.) Partition $R$ into sub-regions $R_{c_1}$ and $R_{c_2}$ (vertically if $d$ is even, horizontally otherwise) proportionally to the number of probes in $\mathcal{P}_{c_1} \cup \mathcal{Q}_{c_1}$ and $\mathcal{P}_{c_2} \cup \mathcal{Q}_{c_2}$
6. (Proceed recursively.) Partition each sub-problem recursively and return:
    a) Return RecursivePartitioning $(d+1, D, R_{c_1}, \mathcal{Q}_{c_1}, \mathcal{P}_{c_1})$
       $\cup$ RecursivePartitioning $(d+1, D, R_{c_2}, \mathcal{Q}_{c_2}, \mathcal{P}_{c_2})$
```

most (if not all) cycles of the deposition sequence.

Once the pivot candidates are selected, the main recursive procedure is called (Algorithm 2). The output of this procedure is a set of assignments $\mathcal{A} = \{a_1, a_2, \dots a_{2^D}\}$, where each $a_i = (\mathcal{P}_i \cup \mathcal{Q}_i, R_i)$, i.e., a_i consists of a set of probes (pivots and non-pivots) and a defined sub-region R_i of the chip. Each assignment can then be processed, independently, by a placement algorithm.

At Step 2 of Algorithm 2, a pair of pivots q_{c_1} and $q_{c_2} \in \mathcal{Q}$ is selected such that the conflict index distance between their embeddings $C(c_1, c_2)$ is maximal; in case of BLM, the Hamming distance $H(c_1, c_2)$ is used. Instead of checking every possible pair of pivots, the following heuristic is applied: First, a probe index c_1 is randomly selected from $\{1, \dots, |\mathcal{Q}|\}$. Then, a probe index $c_2 \neq c_1$ maximizing $C(c_2, c_1)$ is selected. This procedure is repeated for a fixed number of times, and the pair with maximum $H(c_1, c_2)$ is used in this iteration.

Step 3 partitions the set of pivot candidates $\mathcal{Q}$ into sub-sets $\mathcal{Q}_{c_1}$ and $\mathcal{Q}_{c_2}$ associated with pivots q_{c_1} and q_{c_2}, respectivelly. This is done by comparing each of the remaining pivot candidates $q_k \in \mathcal{Q}$ with q_{c_1} and q_{c_2} and assigning it to the sub-set $\mathcal{Q}_{c_j}$ whose pivot results in minimum $C(k, c_j)$ over $j = 1, 2$, or minimum $H(k, c_j)$ in case of BLM.

A similar approach is used to partition the set of non-pivot probes $\mathcal{P}$ into sub-sets $\mathcal{P}_{c_1}$ and $\mathcal{P}_{c_2}$ (Step 4). The difference is that a non-pivot probe p_k is assigned to a sub-set $\mathcal{P}_{c_j}$ considering all valid embeddings of p_k with respect to the embedding of pivot q_{c_j}. This is done by computing the *minimum conflict index distance* $M_C(k, c_j)$ or the *minimum Hamming distance* $M_H(k, c_j)$ in case of BLM. $M_C(k, c_j)$ is defined as the minimum conflict index distance $C(\bar{k}, c_j)$ between any embedding $\varepsilon_{\bar{k}}$ of p_k and a fixed embedding ε_{c_j} (see Section 2.3 for the definition of conflict index distance). Similarly, $M_H(k, c_j)$ is defined as the minimum Hamming distance $H(\bar{k}, c_j)$ between any embedding $\varepsilon_{\bar{k}}$ of p_k and ε_{c_j} (see Section 2.2 for the definition of Hamming distance).

$M_C(k, c_j)$ and $M_H(k, c_j)$ are computed with the OSPE algorithm of Section 5.1. However, since at this point the probes have not yet been assigned to spots, we use a variant of OSPE that ignores the location of the probes (and thus the distance-dependent weights γ) by setting the U_t and $M_{i,t}$ costs (Equations 5.1 and 5.2), in the CIM case, as follows:

$$U_t := \mathbb{1}_{\{\varepsilon_{c_j,t}=0\}} \cdot \omega(\varepsilon_{c_j}, t),$$
$$M_{i,t} := c \cdot \exp(\theta \cdot (1 + \min\{i, \ell - i\})) \cdot \mathbb{1}_{\{\varepsilon_{c_j,t}=1\}}.$$

At Step 5, the region R is partitioned into sub-regions R_{c_1} and R_{c_2} proportionally to the number of probes in $\mathcal{P}_{c_1} \cup \mathcal{Q}_{c_1}$ and $\mathcal{P}_{c_2} \cup \mathcal{Q}_{c_2}$. The algorithm alternates between vertical (if current partitiong depth d is even) and horizontal (if d is odd) divisions.

Pivot Partitioning continues recursively up to a pre-defined maximum partitioning depth D. When $d = D$, it returns an assignment of all probes of $\mathcal{P} \cup \mathcal{Q}$ (pivots and non-pivots) to region R (Step 1). Before that, however, the algorithm re-embeds each probe $p_k \in \mathcal{P}$ optimally with respect to all pivots $q_j \in \mathcal{Q}$ using another variant of OSPE with costs U_t and $M_{i,t}$, in case of CIM, set as follows:

$$U_t := \sum_{q_j \in \mathcal{Q}} \mathbb{1}_{\{\varepsilon_{j,t}=0\}} \cdot \omega(\varepsilon_j, t),$$

$$M_{i,t} := c \cdot \exp(\theta \cdot (1 + \min\{i, \ell - i\})) \cdot \sum_{q_j \in \mathcal{Q}} \mathbb{1}_{\{\varepsilon_{j,t}=1\}}.$$

6.4.1 Results

Table 6.2 shows a comparison between Pivot Partitioning and Centroid-based Quadrisection. For this comparsion, we reproduce the results of Kahng et al. (2003b), which

Table 6.2: Comparison between Pivot Partitioning (PP) and Centroid-based Quadrisection (CQ) on chips containing random probes sequences of length 25 embedded in a 100-step deposition sequence (probes are, initially, synchronously embedded). Chip dimensions range from 100 × 100 to 500 × 500. Partitioning depths vary from $D = 1$ to $D = 3$ for CQ and, equivalently, from $D = 2$ to $D = 6$ for PP. Both partitionings use Row-Epitaxial for the placement with 1-threading and $Q = 20\,000$, and are followed by the Sequential re-embedding algorithm with threshold $W = 0.1\%$. The data shows the normalized border length of chips produced by CQ as reported by Kahng et al. (2003b), and the results of using PP on similar input. The relative diffence between the two algorithms is shown in percentage.

	100 × 100	200 × 200	300 × 300	500 × 500
	NBL	NBL	NBL	NBL
CQ $D = 1$	19.8595	19.1558	19.4735	19.1310
PP $D = 2$	**19.7414**	**18.6572**	**17.9959**	**17.3154**
Relative	-0.60%	-2.60%	-7.59%	-9.49%
CQ $D = 2$	**20.1673**	19.4199	19.0263	18.7480
PP $D = 4$	20.4057	**19.1756**	**18.4533**	**17.6462**
Relative	+1.18%	-1.26%	-3.01%	-5.88%
CQ $D = 3$	**20.7378**	**19.7625**	19.1470	18.6523
PP $D = 6$	21.1305	19.8459	**19.0458**	**18.1701**
Relative	+1.89%	+0.42%	-0.53%	-2.59%

used chips with random probes of length $\ell = 25$ that were, initially, synchronously embedded in a cyclic deposition sequence of length $N = 100$. We run PP on similar input and report the results with equivalent partitioning depths (two levels of PP are equivalent to one level of CQ). Both algorithms were configured for BLM and used 1-threading and Row-Epitaxial for the placement with $Q = 20\,000$. Since PP also modifies the probes' embeddings, we compare the results obtained by both algorithms after a re-embedding phase with Sequential (Section 5.4) using threshold $W = 0.1\%$.

Our results show that PP produced layouts with less border conflicts than CQ except on the smaller chips with higher partitioning depths. On 500×500 chips, for instance, PP with $D = 2$ produced a layout with 9.49% less border conflicts than CQ with $D = 1$, on average. With $D = 6$ (respectively, $D = 3$ for CQ), this difference droped to 2.60%. On 100×100 chips, however, PP produced worse layouts, with up to 1.89% more border conflicts with $D = 6$. We suspect that this disadvantage is due to the "borrowing heuristic" used by CQ (and not implemented in PP) that permits, during placement, borrowing probes from neighboring partitions in order to maintain a high number of probe candidates for filling the last spots of a quadrant.

We also report results of similar experiments using PP and the Greedy placement algorithm compared to using Greedy alone. For these experiments, we used versions of PP and Greedy for border length as well as conflict index minimization (Table 6.3). In all cases, we run the Sequential re-embedding algorithm with threshold $W = 0.1\%$ after placement.

Table 6.3: Normalized border length (NBL) and average conflict index (ACI) of layouts produced by the Greedy placement algorithm and Pivot Partitioning (PP) with varying partitioning depths D on chips containing random probes embedded in a deposition sequence of length 100 (probes are, initially, synchronously embedded). PP uses Greedy for placement inside final regions. In all cases, Greedy uses $Q = 20\,000$ and 0-threading, and placement is followed by a re-embedding phase with Sequential using threshold $W = 0.1\%$. Total time (including partitioning, placement and re-embedding) is reported in seconds.

	200×200		300×300		500×500	
	NBL	Time	NBL	Time	NBL	Time
Greedy	20.7696	173.8	20.2921	560.5	19.5884	2 214.3
PP $D = 2$	18.6572	50.3	17.9959	335.8	17.3154	1 921.2
Relative	-10.17%	-71.0%	-11.32%	-40.1%	-11.60%	-13.2%
PP $D = 4$	19.1756	26.6	18.4533	92.2	17.6462	913.6
Relative	-7.67%	-84.7%	-9.06%	-83.6%	-9.92%	-58.7%
PP $D = 6$	19.8459	23.3	19.0458	60.2	18.1701	254.4
Relative	-4.45%	-86.6%	-6.14%	-89.3%	-7.24%	-88.5%

	200×200		300×300		500×500	
	ACI	Time	ACI	Time	ACI	Time
Greedy	469.6163	1 077.8	454.7646	2 780.5	440.8775	8 151.0
PP $D = 2$	410.9014	533.2	396.1600	1 799.2	380.6258	6 940.4
Relative	-12.50%	-50.5%	-12.89%	-35.3%	-13.67%	-14.9%
PP $D = 4$	426.4966	406.3	409.6784	1 024.0	389.2871	4 505.6
Relative	-9.18%	-62.3%	-9.91%	-63.2%	-11.70%	-44.7%
PP $D = 6$	444.0277	366.1	425.2855	891.5	403.9497	3 038.1
Relative	-5.45%	-66.0%	-6.48%	-67.9%	-8.38%	-62.7%

Our results show that PP improves the quality of layouts in both measures at the same time that it significantly reduces running time. The best layouts were invariably achived with $D = 2$ and the improvements were higher on larger chips. The reduction in normalized border length was up to 11.60% (from 19.5884 to 17.3154) on 500×500 chips with $D = 2$ when compared with no partitioning. In this particular case, there was also a reduction of 13.2% in running time (from 2 214.3 to 1 921.2 seconds). With CIM, the reduction in average conflict index was up to 13.67% (from 440.8775 to 380.6258) on 500×500 chips with $D = 2$ when compared with no partitioning.

Increasing the partitioning depth up to $D = 6$ still resulted in better layouts, although with relatively less reduction in normalized border length and average conflict index when compared to $D = 2$. In terms of running time, however, we observed a reduction of as much as 89.3% in the BLM case (from 560.5 to 60.2 seconds) and 67.9% in the CIM case (from 2 780.5 to 891.5 seconds) on 300×300 chips with $D = 6$ when compared with no partitioning.

It should be noted that the results shown on Tables 6.2 and 6.3 use a deposition sequence of length $T = 100$, which allows a considerable degree of freedom for embedding probes of length $\ell = 25$; these experiments were mainly performed to compare PP

Table 6.4: Normalized border length (NBL) of layouts produced by the Greedy placement algorithm and Pivot Partitioning (PP) with varying partitioning depths D on chips containing random probes embedded in the standard Affymetrix deposition sequence (of length 74; probes are, initially, left-most embedded). PP uses Greedy for placement inside final regions. In all cases, Greedy uses Q as indicated and 0-threading, and placement is followed by a re-embedding phase with Sequential using threshold $W = 0.2\%$. Total time (including partitioning, placement and re-embedding) is reported in seconds.

		300×300		500×500		800×800	
Q	Alg.	NBL	Time	NBL	Time	NBL	Time
5K	Greedy	18.2121	103.3	17.4851	358.4	16.8201	949.4
	PP $D = 2$	18.4376	87.0	17.8102	315.3	17.1683	922.0
	Relative	+1.24%	-15.7%	+1.86%	-12.0%	+2.07%	-2.9%
	PP $D = 4$	18.6193	58.7	17.9299	267.9	17.3763	885.2
	Relative	+2.24%	-43.2%	+2.54%	-25.3%	+3.31%	-6.8%
	PP $D = 6$	19.1262	31.2	18.2090	149.8	17.5295	671.6
	Relative	+5.02%	-69.8%	+4.14%	-58.2%	+4.22%	-29.3%
20K	Greedy	17.9726	582.7	17.2779	2 012.4	16.6258	5 782.1
	PP $D = 2$	18.1954	295.6	17.5494	1 612.5	16.9620	5 083.1
	Relative	+1.24%	-49.3%	+1.57%	-19.9%	+2.02%	-12.1%
	PP $D = 4$	18.6124	61.2	17.7584	696.6	17.1114	3 924.4
	Relative	+3.56%	-89.5%	+2.78%	-65.4%	+2.92%	-32.1%
	PP $D = 6$	19.1262	31.3	18.2083	150.5	17.4450	1 158.8
	Relative	+6.42%	-94.6%	+5.38%	-92.5%	+4.93%	-80.0%

with previous results on CQ. In practice, the production of commercial microarrays is likely to use shorter deposition sequences. Affymetrix chips, for instance, are synthesized in 74 synthesis steps. For this reason, we also show the results of using Pivot Partitioning on chips with random 25-mer probes left-most embedded in the standard Affymetrix deposition sequence. In these experiments we use the Greedy placement algorithm with $Q = 5\,000$ and $Q = 20\,000$, and we report the results of PP compared with layouts produced with no partitioning (using Greedy alone).

With BLM (Table 6.4), we observed that partitioning the chip always resulted in worse layouts than without partitioning, although there was always a reduction in running time. Again, increasing the partitioning depth from $D = 2$ to $D = 6$ worsened the results. For instance, the percentual increase in normalized border length on 800×800 arrays in comparison with no partitioning raised from 2.02% with $D = 2$ to 4.93% with $D = 6$ (with $Q = 20$ K), although the percentual reduction in running time also raised from 12.1% to 80.0%. The reduction in running time was higher on the smaller arrays and with higher values of Q because, in these cases, the restriction on the number of probe candidates per spot is more significant.

With respect to CIM (Table 6.5), however, the partitioning resulted in improved layouts in some cases, especially for the larger chips. With $D = 4$ and $Q = 5$K, we observed a reduction of 4.41% in average conflict index on 800×800 arrays, although

Table 6.5: Average conflict index (ACI) of layouts produced by Greedy and Pivot Partitioning (PP) with varying partitioning depths D on random chips with probes left-most embedded in the Affymetrix deposition sequence. PP uses Greedy for placement inside final regions. In all cases, Greedy uses Q as indicated and 0-threading, and placement is followed by Sequential re- embedding with $W = 0.2\%$. Total time is reported in seconds.

		300×300		500×500		800×800	
Q	Alg.	ACI	Time	ACI	Time	ACI	Time
5K	Greedy	436.8630	511.3	428.7410	1 479.8	422.6277	3 870.0
	PP $D = 2$	432.8319	621.8	419.9128	1 863.0	410.8418	4 865.1
	Relative	-0.92%	+21.6%	-2.06%	+25.9%	-2.79%	+25.7%
	PP $D = 4$	441.2177	510.6	418.1961	1 724.1	403.9992	4 781.8
	Relative	+1.00%	-0.1%	-2.46%	+16.5%	-4.41%	+23.6%
	PP $D = 6$	459.5480	378.7	429.4306	1 356.5	407.4338	4 275.3
	Relative	+5.19%	-25.9%	+0.16%	-8.3%	-3.60%	+10.5%
20K	Greedy	412.5536	2 008.5	398.6096	4 555.5	389.3929	12 535.3
	PP $D = 2$	423.0404	1 184.5	400.7174	4 837.2	386.0881	13 898.2
	Relative	+2.54%	-41.0%	+0.53%	+6.2%	-0.85%	+10.9%
	PP $D = 4$	440.4754	539.6	411.0308	2 940.8	388.3189	11 656.7
	Relative	+6.77%	-73.1%	+3.12%	-35.4%	-0.28%	-7.0%
	PP $D = 6$	459.5725	378.6	428.7111	1 461.4	402.3157	6 629.7
	Relative	+11.40%	-81.2%	+7.55%	-67.9%	+3.32%	-47.1%

that also resulted in an increase of 23.6% in running time. On 500×500 chips, PP with $D = 6$ and Greedy with $Q = 20$K produced, in approximately the same time, a layout that was slightly better than the layout produced by Greedy with $Q = 5K$ and no partitioning (428.7111 ACI in 1 461.4 seconds versus 428.7410 ACI in 1 479.8 seconds, respectively). Only in one case we observed a reduction of running time combined with an improvement in solution quality: On 800×800 arrays, PP with $D = 4$ and Greedy with $Q = 20$K achieved reductions of 0.28% in ACI and 7.0% in running time when compared to Greedy alone.

6.5 Summary

In this chapter we surveyed several partitioning algorithms that are able to break the Microarray Layout Problem into smaller sub-problems; two of these algorithms were previously unpublished: 1-D and 2-D Partitioning.

Our results show that a partitioning can be used to improve solution quality and /or reduce running time. However, several aspects of the problem such as chip size, placement algorithm, type of embeddings, deposition sequence length and type of optimization (BLM or CIM), must be taken into account when choosing the partitioning algorithm and its parameters.

While Centroid-based Quadrisection and Pivot Partition offer more homogeneous improvements over all synthesis steps, 1-DP and 2-DP are able to achieve significant reduction of conflicts for a few selected masks, which can be benefical for the conflict index measure where a conflict in the middle of the probe is penalized more severely.

On chips with a 100-step cyclic deposition sequence, Pivot Partitioning outperformed previous results of CQ on larger chips because the approach of simultaneously re-embedding and assigning probes to regions better exploits the extra freedom on the probes' embeddings provided by the long deposition sequence. We believe that the comparatively worse results achieved by PP on the smaller chips with higher partitioning depths are due to the borrowing heuristic implemented in CQ that allows the placement algorithm to keep a high number of probe candidates per spot when the last sites of a quadrant are being filled.

With shorter deposition sequences, we have showed that the restriction in number of candidates per probe during placement of the last spots of a region often impacts the solution quality more significantly than the gains due to grouping similar probes together. As a result, in terms of BLM, PP failed to improve the quality of layouts produced by the Greedy placement algorithm. In terms of CIM, however, PP was able to reduce running time as well as ACI, probably because there is more room for optimization in this measure. Again, the borrowing heuristic implemented in CQ could improve the results of PP in both measures. It should be noted, however, that the effects of a partitioning on Greedy and Row-Epitaxial are mainly due to a particularity of their placement strategies; other placement algorithms such as Sliding-Window Matching (Section 3.4) are not expected to be impaired in the same way.

Chapter 7

Merging Placement and Re-embedding

The problem with the traditional “place and re-embed” approach is that the arrangement of probes on the chip is decided based on embeddings that are likely to change during the re-embedding phase. Intuitively, better results should be obtained when the placement and embedding phases are considered simultaneously instead of separately. However, because of the generally high number of embeddings of each single probe, it is not easy to design algorithms that efficiently use the additional freedom and run reasonably fast in practice.

We describe Greedy+, the first placement algorithm that simultaneously places and re-embeds the probes, and compare it with Row-Epitaxial, the best known large-scale placement algorithm.

7.1 Greedy+

The goal is to design an algorithm that is similar to Row-Epitaxial, so that we can make a better assessment of the gains resulting from merging the placement and re-embedding phases.

Greedy+ fills the spots row-by-row, from left to right, in a greedy fashion, similarly to Row-Epitaxial. Also, for each spot s, it looks at Q probe candidates and chooses the one that can be placed at s with minimum cost. The difference is that we now consider all possible embeddings of a candidate p instead of only p's initial embedding. This is done by temporarily placing p at s and computing its optimal embedding with respect to the already-filled neighbors of s (using OSPE from Section 5.1).

Compared to Row-Epitaxial, Greedy+ spends more time evaluating each probe candidate p for a spot s. While Row-Epitaxial takes $O(T)$ time to compute the conflict index or the border length resulting from placing p at s, Greedy+ requires $O(\ell T)$ time since it uses OSPE (recall that ℓ is the probe length and T is the length of the deposition sequence). To achieve a running time comparable to Row-Epitaxial, we must therefore consider lower candidate numbers Q.

There are a few optimizations that reduce the time spent with OSPE computations when several probe candidates are examined in succession for the same spot. First, we note that if two probe candidates p and p' share a common prefix of length l, the first $l+1$ rows of the OSPE dynamic programming matrix D will be identical. In other words, if we have calculated the minimum cost of p, we can speed up the calculation of the minimum cost of p' by skipping the first $l+1$ rows of D.

In order to fully exploit this fact, we examine the probes in lexicographical order so that we maximize the length of the common prefix between two consecutive candidates. We keep a doubly-linked list of probes and remove a probe p from the list when it is placed. For the next spot to be filled, we look at Q probes in the list around p's former position, e.g., at $Q/2$ probes to the left and to the right of p.

Second, the U_t costs of OSPE need to be computed only once for a given spot s since U_t does not depend on the probe placed at s. Thus, in order to examine another candidate, we only need to recompute the $M_{i,t}$ costs.

Finally, once we know that a probe candidate p can be placed at s with minimum cost C, we can stop the OSPE computation for another candidate p' as soon as all values in a row of D are greater than or equal to C.

7.2 Results

We compare the results of Greedy+ with Row-Epitaxial. To be fair, since Row-Epitaxial is a traditional placement algorithm that does not change the embeddings, we need to compare the layouts obtained by both algorithms after a re-embedding phase. For this task we use the Sequential algorithm (Section 5.4) with thresholds of $W = 0.1\%$ for border length minimization and $W = 0.5\%$ for conflict index minimization, so that the algorithm stops as soon as the improvement in one iteration drops below W.

Table shows the total border length and the average conflict index of layouts produced by both algorithms on two chips with dimensions 335×335 and 515×515, filled with probes randomly selected from existing GeneChip arrays (E.Coli Genome 2.0 and Maize Genome, respectively). Probes are initially left-most embedded into the standard 74-step Affymetrix deposition sequence $\{\text{TGCA}\}^{18}$TG. The parameter Q is chosen differently for both algorithms so that the running time is approximately comparable (e.g., for border length minimization, $Q = 350$ for Greedy+ corresponds to $Q = 10\,000$ for Row-Epitaxial). We make the following observations.

Table 7.1: Normalized border length (NBL) of layouts produced by Greedy+ on random chips of various dimensions. The amplitude of the k-threading and the number Q of candidades per spot are shown. All results are averages over a set of five arrays and running times are reported in minutes.

		$Q = 500$		$Q = 1\,000$		$Q = 2\,000$	
Dim.	k	NBL	Time	NBL	Time	NBL	Time
300×300	0	17.9356	5.4	17.7136	10.6	17.5460	20.6
	1	18.0922	5.4	17.8988	10.5	17.7501	20.4
	2	17.9886	5.4	17.7905	10.5	17.6342	20.5
	3	17.9339	5.7	17.7406	10.5	17.5799	20.5
	4	17.8978	5.7	17.7155	11.1	17.5506	20.5
	5	17.8862	5.7	17.7013	10.6	17.5359	20.5
	6	17.8749	5.4	17.6908	10.6	17.5225	20.5
	7	17.8641	5.5	17.6807	10.6	17.5223	20.6
	8	17.8605	5.4	17.6711	10.6	17.5141	20.6
	9	17.8519	5.4	17.6685	10.6	17.5083	20.6
	10	17.8518	5.4	17.6657	10.6	17.5067	20.6
	11	**17.8427**	5.5	17.6705	10.6	**17.5066**	20.6
	12	17.8431	5.4	**17.6643**	10.6	17.5070	20.6
500×500	0	17.3240	14.9	17.0576	29.1	17.0761	57.0
	1	17.4648	14.8	17.2483	28.9	16.9650	56.5
	2	17.3372	14.9	17.1318	29.0	16.9135	56.4
	3	17.2732	14.9	17.0785	29.0	16.8855	56.5
	4	17.2371	14.9	17.0436	29.0	16.8676	56.8
	5	17.2143	14.9	17.0264	29.3	16.8557	57.2
	6	17.1990	15.0	17.0141	29.3	16.8420	57.2
	7	17.1812	15.0	17.0049	29.3	16.8398	57.2
	8	17.1774	15.0	16.9965	29.3	16.8346	57.0
	9	17.1704	15.0	16.9921	29.4	16.8332	57.3
	10	17.1666	15.8	16.9876	29.2	16.8294	59.7
	11	17.1629	15.0	**16.9814**	29.1	16.8280	56.8
	12	**17.1594**	14.9	16.9821	29.3	**16.7983**	56.7
800×800	0	16.7983	38.0	16.4944	73.8	16.2640	144.4
	1	16.8849	37.7	16.6615	73.3	16.4780	143.3
	2	16.7420	37.8	16.5377	73.5	16.3626	143.6
	3	16.6693	37.9	16.4775	73.9	16.3070	143.9
	4	16.6266	38.0	16.4375	73.8	16.2707	144.2
	5	16.5938	38.1	16.4096	74.2	16.2497	145.1
	6	16.5700	38.2	16.3919	74.3	16.2334	145.2
	7	16.5543	38.2	16.3801	74.6	16.2237	145.2
	8	16.5435	38.1	16.3691	74.5	16.2171	145.3
	9	16.5379	38.2	16.3646	74.7	16.2115	145.8
	10	16.5297	38.0	16.3586	74.0	16.2094	144.5
	11	16.5229	38.0	16.3539	74.0	16.2039	144.5
	12	**16.5210**	38.2	**16.3518**	74.1	**16.2022**	144.6

First, increasing Q linearly increases placement time, while only marginally improving chip quality for border length minimization.

Second, re-embedding runs very quickly for border length minimization, even on the larger chip. For conflict index minimization, the time for the re-embedding phase exceeds the time for the placement phase for both algorithms.

Finally, Greedy+ always produces better layouts in the same amount of time (or less) while looking at fewer probe candidates. In particular, for conflict index minimization on the 515×515 chip with $Q = 5\,000$ resp. 200, Greedy+ and Sequential improve the average conflict index by 18% (from 554.87 to 454.84) and need only 60% of the time, compared to Row-Epitaxial and Sequential.

Table 7.2: Average conflict index (ACI) of layouts produced by Greedy+ on random chips of various dimensions. The amplitude of the k-threading and the number Q of candidades per spot are shown. All results are averages over a set of five arrays and running times are reported in minutes.

		$Q = 500$		$Q = 1\,000$		$Q = 2\,000$	
Dim.	k	ACI	Time	ACI	Time	ACI	Time
300×300	0	**462.3882**	5.8	**443.3786**	10.5	**425.9132**	19.8
	1	468.6485	5.8	449.1931	10.6	431.1021	19.9
	2	472.3753	5.8	452.5054	10.6	434.1209	19.9
	3	474.3210	5.8	454.6870	10.6	436.2880	20.0
	4	474.2031	5.8	454.6782	10.6	436.2529	19.9
500×500	0	**457.3329**	15.8	**437.3920**	28.8	**419.2114**	54.2
	1	463.6259	16.0	443.7018	30.4	424.5009	54.7
	2	467.3461	15.9	447.5021	29.0	428.3882	54.8
	3	469.2554	16.6	449.4136	29.1	430.4992	55.0
	4	468.9371	16.0	449.5197	29.1	430.4662	58.0
800×800	0	**451.8074**	40.0	**431.8977**	73.0	**413.3451**	144.3
	1	458.1598	40.3	437.8440	73.5	418.9562	138.4
	2	461.6418	40.3	441.6484	73.3	423.0075	145.9
	3	463.5349	40.3	443.7868	73.6	425.2302	138.9
	4	463.1225	40.3	443.7802	73.7	425.3695	139.0

Table 7.3: Normalized border length (NBL) of layouts produced by Greedy and Greedy+ on random chips with the number Q of candidades per spot set in such a way that both algorithms spend approximately the same amount of time. Total time (including placement and re-embedding) is reported in minutes. Both algorithms use 0-threading and are followed by two passes of re-embedding optimization with Sequential. The relative difference in NBL between the two approaches is shown in percentage.

	Greedy and Sequential			Greedy+ and Sequential			
Dim.	Q	NBL	Time	Q	NBL	Time	Relative
300×300	10 000	18.0900	4.4	390	**17.8717**	4.4	1.21%
	20 000	17.9726	9.0	820	**17.6236**	8.8	1.94%
500×500	10 000	17.3809	15.4	510	**17.1662**	15.4	1.24%
	20 000	17.2779	32.4	1 100	**16.8901**	32.2	2.24%
800×800	10 000	16.7143	43.8	570	**16.5945**	43.4	0.72%
	20 000	16.6258	93.2	1 260	**16.2799**	92.9	2.08%

Table 7.4: Average conflict index (ACI) of layouts produced by Greedy and Greedy+ (with 0-threading) on random chips in approximately the same amount of time (total time in minutes including two passes of Sequential re-embedding optimization). The relative difference in ACI between the two approaches is shown in percentage.

	Greedy and Sequential			Greedy+ and Sequential			
Dim.	Q	ACI	Time	Q	ACI	Time	Relative
300 × 300	10 000	**423.1330**	13.9	1 070	438.4015	14.0	-3.61%
	20 000	**412.5536**	24.1	2 180	420.8863	24.2	-2.02%
	80 000	402.4365	54.3	5 500	**401.7005**	54.0	0.18%
500 × 500	10 000	**412.5468**	43.2	1 225	428.5082	43.7	-3.87%
	20 000	**398.6096**	77.0	2 580	409.6446	76.9	-2.77%
	140 000	375.5428	352.2	13 500	**374.9914**	359.9	0.15%
800 × 800	10 000	**405.3133**	113.9	1 315	421.2380	113.7	-3.93%
	20 000	**389.3929**	207.9	2 790	401.7969	208.5	-3.19%
	270 000	000.0000	1 000.0	30 000	**000.0000**	1 000.0	0.00%

Chapter 8

Analysis of Affymetrix Microarrays

General physical structure of GeneChip arrays. Control and special probes, checkerboard patterns on the borders, empty spots.

Probe pairs: perfect match (PM) and mismatch (MM) probes. Rows of PM and MM probes on the chip.

As we mentioned in Chaper 8, all GeneChip arrays that we know of can be asynchronously synthesized in 74 steps with the standard Affymetrix deposition sequence (18.5 cycles of TGCA).

This suggests that only sub-sequences of this sequence can be used as probes on Affymetrix chips. Rahmann (2006) shows that this covers about 98.45% of all 25-mers, however, it seems that Affymetrix uses an even more restrictive probe selection criterion. This is because GeneChip probes always appear in pairs, with the perfect match (PM) and the mismatch (MM) probes being located next to each other (in alternating rows of PM and MM probes), and there is evidence that the embeddings of these probes are aligned in such a way that only the middle bases are not aligned.

Chapter 9

Shortest Common Supersequence

Chapter 10

Discussion

This thesis makes several contributions to the field of....

We introduce the *conflict index...*

In Chapter X we present a algorithm...

We are not aware of any previous work that combines placement and re-embedding, even our results showed that such an approach has obvious benefits...

We have surveyed algorithms for the microarray layout problem (MLP), divided into placement, (re-)embedding, and partitioning algorithms. Because of the super-exponential number of possible layouts and the relation to the quadratic assignment problem (QAP), we cannot expect to find optimal solutions. Indeed, the algorithms we present are heuristics with an emphasis on good scalability and ideally a user-controllable trade-off between running time and solution quality, albeit without any known provable guarantees.

We believe that solution quality is more important than the running time of a placement algorithm. Even if an algorithm takes a couple of days to complete, it is time well spent given that commercial microarrays are likely to be produced in large quantities. This is specially true when we consider the time required for the whole design process of a microarray chip. Even customer designed chips, that usually have a limited number of produced units, are likely to benefit from a few extra hours of computing time.

Among the presented approaches, two recent ones (Pivot Partitioning and Greedy+) indicate that the traditional "place first and then re-embed" approach can be improved upon by merging the partitioning/placement and (re-)embedding phases. Ongoing work will show the full potential of such combined approaches.

As a suggestion for further work, we note the needed for improving the selection of probe candidates considered for filling each spot. For example, instead using a sorted list of probes, one could use a TSP tour like the one described in Section 3.2. However, it is not clear if the more time-consuming TSP approach will pay off (instead, we could use this extra time to look at more candidates).

An alternative that sounds interesting would be to build some kind of “clustering” of the probes, perhaps based on a graph or a tree, in such a way that we can find similar probes more easily and spend time on candidates that are more likely to produce less conflicts.

Question: why PM and MM probes are placed together and aligned except for the middle base?

Bibliography

W. Adams, M. Guignard, P. Hahn, and W. Hightower. A level-2 reformulation-linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*, To appear.

H. Binder and S. Preibisch. Specific and nonspecific hybridization of oligonucleotide probes on microarrays. *Biophys J*, 89(1):337–352, Jul 2005. doi: 10.1529/biophysj.104.055343. URL `http://dx.doi.org/10.1529/biophysj.104.055343`.

E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms.* Kluwer Academic Publishers, 1997.

P. Chase. Subsequence numbers and logarithmic concavity. *Discrete Mathematics*, 16: 123–140, 1976.

S. A. de Carvalho Jr. and S. Rahmann. Searching for the shortest common supersequence. Technical Report 2005-03, Technische Fakultät der Universität Bielefeld, Apr 2005.

S. A. de Carvalho Jr. and S. Rahmann. Improving the layout of oligonucleotide microarrays: Pivot Partitioning. In P. Bucher and B. Moret, editors, *Proceedings of the 6th Workshop of Algorithms in Bioinformatics*, volume 4175 of *Lecture Notes in Computer Science*, pages 321–332. Springer, 2006a. doi: 10.1007/11851561. URL `http://www.springerlink.com/content/h9r4n673058032xm`.

S. A. de Carvalho Jr. and S. Rahmann. Microarray layout as a quadratic assignment problem. In D. Huson, O. Kohlbacher, A. Lupas, K. Nieselt, and A. Zell, editors, *Proceedings of the German Conference on Bioinformatics*, volume P-83 of *Lecture Notes in Informatics (LNI)*, pages 11–20. Gesellschaft für Informatik, 2006b.

S. A. de Carvalho Jr. and S. Rahmann. Modeling and optimizing oligonucleotide microarray layout. In I. Mandoiu and A. Zelikowsky, editors, *Bioinformatics Algorithms: Techniques and Applications*, Wiley Book Series on Bioinformatics. Wiley, 2007. To appear.

W. Feldman and P. Pevzner. Gray code masks for sequencing by hybridization. *Genomics*, 23(1):233–235, 1994. doi: 10.1006/geno.1994.1482. URL `http://dx.doi.org/10.1006/geno.1994.1482`.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

S. P. Fodor, J. L. Read, M. C. Pirrung, L. Stryer, A. T. Lu, and D. Solas. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251(4995):767–773, 1991.

S. P. Fodor, R. P. Rava, X. C. Huang, A. C. Pease, C. P. Holmes, and C. L. Adams. Multiplexed biochemical assays with biological chips. *Nature*, 364(6437):555–556, Aug 1993. doi: 10.1038/364555a0. URL `http://dx.doi.org/10.1038/364555a0`.

H. Gabow. An efficient implementation of Edmond's algorithm for maximum matching on graphs. *J. ACM*, 23:221–234, 1976.

J. L. Gross and J. Yellen, editors. *Handbook of graph theory.* CRC Press, 2004.

P. Hahn, T. Grant, and N. Hall. A branch-and-bound algorithm for the quadratic assignment problem based on the hungarian method. *European Journal of Operational Research*, 108:629–640(12), 1998. doi: doi:10.1016/S0377-2217(97)00063-5. URL `http://www.ingentaconnect.com/content/els/03772217/1998/00000108/00000003/art00063`.

S. Hannenhalli, E. Hubell, R. Lipshutz, and P. A. Pevzner. Combinatorial algorithms for design of DNA arrays. *Adv Biochem Eng Biotechnol*, 77:1–19, 2002.

E. A. Hubbell, M. S. Morris, and J. L. Winkler. Computer-aided engineering system for design of sequence arrays and lithographic masks. United States Patent number 5,856,101, Jan 1999.

A. Kahng, I. Mandoiu, P. Pevzner, S. Reda, and A. Zelikovsky. Border length minimization in DNA array design. In R. Guigó and D. Gusfield, editors, *Algorithms in Bioinformatics (Proceedings of WABI)*, volume 2452 of *Lecture Notes in Computer Science*, pages 435–448. Springer, 2002. URL `http://www.springerlink.com/content/pqp7c7emyk7gmx3u`.

A. B. Kahng, I. Mandoiu, P. Pevzner, S. Reda, and A. Zelikovsky. Engineering a scalable placement heuristic for DNA probe arrays. In *Proceedings of the seventh annual international conference on research in computational molecular biology (RECOMB)*, pages 148–156. ACM Press, 2003a. doi: http://doi.acm.org/10.1145/640075.640095.

A. B. Kahng, I. Mandoiu, S. Reda, X. Xu, and A. Z. Zelikovsky. Evaluation of placement techniques for DNA probe array layout. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 262–269. IEEE Computer Society, 2003b. doi: http://dx.doi.org/10.1109/ICCAD.2003.65.

T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC Press, 1999.

Y. Li, P. Pardalos, and M. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994. URL `http://www.research.att.com/~mgcr/doc/grpqap.ps.Z`.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

C. A. S. Oliveira, P. M. Pardalos, and M. G. C. Resende. GRASP with path-relinking for the quadratic assignment problem. In C. C. Ribeiro and S. L. Martins, editors, *Proc. of Third Workshop on Efficient and Experimental Algorithms (WEA04)*, volume 3059 of *Lecture Notes in Computer Science*, pages 356–368. Springer-Verlag, 2004.

S. Rahmann. The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, 19(Suppl 2):ii156–ii161, Oct 2003.

S. Rahmann. Subsequence combinatorics and applications to microarray production, DNA sequencing and chaining algorithms. In M. Lewenstein and G. Valiente, editors, *Combinatorial Pattern Matching (CPM)*, volume 4009 of *LNCS*, pages 153–164, 2006.

C. Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39(4):605–629, 1997.

S. Singh-Gasson, R. D. Green, Y. Yue, C. Nelson, F. Blattner, M. R. Sussman, and F. Cerrina. Maskless fabrication of light-directed oligonucleotide microarrays using a digital micromirror array. *Nat Biotechnol*, 17(10):974–978, Oct 1999. doi: 10.1038/13664. URL `http://dx.doi.org/10.1038/13664`.