TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER’S THESIS

Cell spreading for detailed placement

by
Bing Gu

Supervisor:
Prof.Dr. Wil H.A. Schilders (TUE)
Dr.Ir. C.A.J. van Eijk (MAGMA)
Dr. Menno Ewout Verbeek (MAGMA)

Eindhoven, The Netherlands
August 2005
Abstract

The topic of this thesis is cell spreading for detailed placement. Cells, or modules, are the objects to be placed for the placement. Cell placement is the sub-task of the VLSI circuit design that involves arranging the cells on the chip area such that the layout is routable and the overall area of the chip is minimum. At the beginning of placement, all cells are placed randomly in the placement area and there’s only electricity connection relation between cells. Traditionally, common approaches to the placement problem involve 2 steps: global placement (minimizing wire length while spreading cells, but ignoring overlapping) and detailed placement (removing all overlap and appointing cells legal position). We introduce a new step “cell spreading” in between these 2 steps to spread cell more evenly over the placement area. It will be the preparation for the detailed placement. Three cell spreading algorithms are discussed in this thesis for this purpose.

The background of the cell spreading problem is introduced in chapter 1. In chapter 2, we give the possible mathematical notations and formulae. In chapter 3,4,5, we focus on three cell spreading algorithms. The evaluation on each algorithm is based on the test data. The comparison experiment on these three algorithms is given in chapter 6.
Acknowledgement

This thesis is the report of the final project of my master program in computational science and engineering, it also comes at the conclusion of two wonderful years spent in Eindhoven. This period has contributed greatly to my professional and personal growth. I enjoy accepting fresh idea and sharing experience with a lot of people in this dynamic university.

I would like to start my thanks to Prof. Dr. W.H.A. Schilders for his considerate guidance. He gave me many incisive suggestions in this project.

A special appreciation goes to Prof. Dr. R.M.M. Matheij, for giving me the opportunity of being a master student of the CASA group.

I would like to thank my two supervisors in Magma Design Automation. Dr. Ir. Koen van Eijk gave me many suggestions with regards to literatures about the real chip physical design during the course of this project. I am extremely thankful to Dr. Menno Ewout Verbeek. The many discussions I had with him, helped me overcome the problems I faced and helped me understand the intricacies of the algorithms.

Many other people in mathematics and computer science department I would also like to mention here. I would like to say thanks to Dr. Jos Maubach for his instructions and suggestions at the beginning of this project; Dr. Martijn Anthonissen, my course director for arranging all my courses, and he’s always kind to answer my various questions; Dr. Paul de Hass taught me a lot about the way of thinking in programming. Special thanks to lovely Enna Van Dijk, for her kindness and patience in helping me a lot for the practical things.

I would like to thank my friends in Eindhoven. I am so happy to experience these wonderful years with them. Among them, I have to mention my very nice Chinese classmates: Hong Deng, Mary, Lewie, Tim for their company in countless lunches and dinners, and let me still feel warm when I am far away from our mother country.

I can’t end my acknowledgement without mentioning my family. I deeply want to say thanks to my parents and my sister, they always give me much freedom to choose what I like to do. Wherever I’ve been, they always stand by me.
## Contents

1 Introduction 1
   1.1 Microelectronics and VLSI 1
   1.2 VLSI placement 3
   1.3 Cell spreading 5
   1.4 Outline 6

2 Fundamental mathematical notations and the placement model 7
   2.1 Initial situation 7
   2.2 Notations 8
   2.3 Goal 9
   2.4 Typical situations 10
   2.5 Standard cell placement simulated model 11
   2.6 Density of each window 12
   2.7 The cell spreading tool 13
      2.7.1 The framework of the cell spreading tool 13
      2.7.2 Testing measurements 15
      2.7.3 Iterative scheme of the cell spreading tool 16
   2.8 Conclusions 16

3 Pair-movement algorithm 18
   3.1 Overview of the algorithm 18
   3.2 Implementation details 19
   3.3 Experimental results 20
   3.4 Conclusions 24

4 Force-based algorithm with an electrostatic model 27
   4.1 The force for cell spreading 27
   4.2 The mathematical model 28
   4.3 The numerical solution 30
      4.3.1 Discretization 30
      4.3.2 The singular linear system 34
      4.3.3 The gradient of the potential 35
   4.4 Examples of application 37
      4.4.1 The cell’s movement under the force 37
      4.4.2 Experiments 38
   4.5 Conclusions 40

5 Diffusion-based placement algorithm 45
   5.1 Cell spreading and diffusion process 45
   5.2 Equations of the diffusion process 46
   5.3 Numerical simulation of diffusion process 47
      5.3.1 Discretization 47
      5.3.2 Velocity Interpolation 48
      5.3.3 The magnitude factor 49
5.3.4 The smoothed density ........................................ 50
5.4 Experiments .................................................. 50
  5.4.1 Small cases in the diffusion process ................. 50
  5.4.2 Application examples .................................. 51
5.5 Conclusions .................................................. 54

6 Comparison experiments ........................................ 59
  6.1 The testgroups ............................................. 59
  6.2 Experimental Results .................................... 61
  6.3 Complexity analysis ..................................... 62
  6.4 Conclusions ................................................. 63

7 Summary ....................................................... 67
List of Figures

1.1 The VLSI design cycle and the physical synthesis flow .... 2
1.2 A standard-cell layout (local part) ................. 3
1.3 Cells, pins and nets in a standard cell layout ............ 4
1.4 The trajectory of placement design .................. 5
2.1 The initial situation of the placement ............... 7
2.2 The initial placement of the standard cell placement simulated model .......................... 12
2.3 Partition of the placement domain for windows .......... 12
2.4 The framework of the cell spreading tool .............. 13
3.1 The movement for 2 overlapping cells ............... 18
3.2 The overlapping cells ................................ 19
3.3 The initial placement and the initial density distribution (in chapter 3) .................. 21
3.4 The maximum scaled density of window with different $\alpha$ (in chapter 3) ............. 22
3.5 The even distribution measurement with different magnitude $\alpha$ (in chapter 3) ........ 23
3.6 The movement distance with different magnitude factor $\alpha$ (in chapter 3) .......... 24
3.7 The result placement with different magnitude factor (in chapter 3) .................. 25
3.8 The movement distance with different magnitude factor $\alpha$ (in chapter 3) .......... 26
4.1 The grids of the domain ............................. 30
4.2 The meshes for calculating density .................. 31
4.3 The sparsity structure of the matrix A ................ 33
4.4 The gradient of potential of one cell .................. 35
4.5 The linear interpolation .............................. 36
4.6 The state of the cell on the boundary ................ 37
4.7 The result placement: equilibrium state ............. 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8</td>
<td>The initial placement and the initial potential distribution (in chapter 4)</td>
</tr>
<tr>
<td>4.9</td>
<td>The maximum scaled density of window in each iteration (in chapter 4)</td>
</tr>
<tr>
<td>4.10</td>
<td>The even distribution measurement in each iteration (in chapter 4)</td>
</tr>
<tr>
<td>4.11</td>
<td>The movement distance in each iteration (in chapter 4)</td>
</tr>
<tr>
<td>4.12</td>
<td>The result placement with different magnitude factor (in chapter 4)</td>
</tr>
<tr>
<td>4.13</td>
<td>The result potential distribution</td>
</tr>
<tr>
<td>4.14</td>
<td>The experiment with $\alpha = 1.5$ (in chapter 4)</td>
</tr>
<tr>
<td>5.1</td>
<td>Velocity interpolation for one cell</td>
</tr>
<tr>
<td>5.2</td>
<td>The migration trajectory of 1 cell placement in chapter 5</td>
</tr>
<tr>
<td>5.3</td>
<td>The migration trajectory of 20 cell placement in chapter 5</td>
</tr>
<tr>
<td>5.4</td>
<td>The initial placement (in chapter 5)</td>
</tr>
<tr>
<td>5.5</td>
<td>The maximum scaled density of window in each iteration. $\alpha' = \frac{2}{(N^2)}$: in non-smoothed test example</td>
</tr>
<tr>
<td>5.6</td>
<td>The even distribution measurement with different magnitude in each iteration. $\alpha' = \frac{2}{(N^2)}$: in non-smoothed test example</td>
</tr>
<tr>
<td>5.7</td>
<td>The movement distance with different magnitude factor. $\alpha' = \frac{2}{(N^2)}$: in non-smoothed test example</td>
</tr>
<tr>
<td>5.8</td>
<td>The result placement with different $\alpha$ (in chapter 5)</td>
</tr>
<tr>
<td>5.9</td>
<td>The density distribution with different $\alpha$ (in chapter 5)</td>
</tr>
<tr>
<td>6.1</td>
<td>The initial placement of the test case with $n = 1600$</td>
</tr>
<tr>
<td>6.2</td>
<td>Comparison results with algorithm 2 and algorithm 3</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison of result figures ($n = 1600$)</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Two steps in placement</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Iterative scheme of the cell spreading tool</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Parameters of the testing model in chapter 3</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>The results of the experiment example in chapter 3</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Parameters of the testing model in chapter 4</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>The results of the experiment example in chapter 4</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>Parameters of the testing model in chapter 5</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>The results of experiment example ($\alpha = D\Delta t$, $D = 1$)</td>
<td>52</td>
</tr>
<tr>
<td>6.1</td>
<td>The test groups of the initial placement</td>
<td>60</td>
</tr>
<tr>
<td>6.2</td>
<td>The input parameters of algorithms ($\alpha$ is the magnitude factor which is used to control the speed of cells' movement. $N$ is the number of grid cells on each dimension for algorithm 2 and algorithm 3 when we discretize the placement domain)</td>
<td>60</td>
</tr>
<tr>
<td>6.3</td>
<td>The results of the comparison experiments</td>
<td>61</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparison summary of three cell spreading algorithms</td>
<td>63</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this chapter, the background of the cell placement problem in VLSI design is briefly introduced. Common approaches to the placement problem involve two steps: global placement and detailed placement. Here we introduce cell spreading—a new step in between these two steps.

1.1 Microelectronics and VLSI

Microelectronics has been the enabling technology for the development of hardware in the recent decades. The continuously increasing level of integration of electronic devices has led to the fabrication of increasingly complex systems. The integrated circuit technology, based on the use of semiconductor materials, has progressed tremendously. In the late 1980s, circuits with over one million devices have been successfully manufactured. Such circuits are often called Very Large Scale Integration or microelectronic circuits.

Designing increasingly complex circuits requires a larger and larger capital investment, due to the cost of refining the precision of the manufacturing process so that finer and finer devices can be implemented. Similarly, the size of the circuits becomes smaller and smaller, which requires larger efforts in achieving zero-defect designs. A particular feature of microelectronics is the near impossibility to repair integrated circuits, which raises the importance of designing circuits correctly and limiting the manufacturing defects. About the technologies in designing VLSI circuits, we refer the interested reader to the books by Micheli[1] and Veendrick[2].

The VLSI design cycle starts with a formal specification of a VLSI chip, followed by a series of steps, and eventually produces a package chip. A typical design cycle may be represented by the flow chart shown in Figure 1.1. Our

1Very Large Scale Integration: A term describing semiconductor integrated circuits composed of hundreds of thousands of logic elements or memory cells.
emphasis is on the placement which is in the physical synthesis step. Before the physical synthesis step is the Logic synthesis which is usually expressed in a de-

![Diagram of VLSI design cycle and physical synthesis flow]

Figure 1.1: The VLSI design cycle and the physical synthesis flow
tailed circuit diagram. This diagram shows the circuit elements (cells, macros, gates, transistors) and the interconnections between these elements. This representation is also called a netlist. In many cases, a netlist can be created automatically from a logic description by using logic synthesis tools. According to a netlist, a chip may contain several million transistors. Due to the limitations of memory space and computation power available it may not be possible to layout the entire chip in the same step. Therefore, the chip is normally partitioned into sub-chips, called blocks. In the work of floorplanning, the area of each block can be estimated after partitioning and is based approximately on the number and the type of components in that block. Completing the interconnections between blocks according to the specified netlist will be done in routing. The details of the VLSI design cycle can be found in [3].
1.2 VLSI placement

Cell placement is the subtask of the VLSI circuit design process that involves arranging the cells on the chip area such that the layout is routable and the overall area of the chip is minimal. Other objectives include minimum delay, minimum power consumption and minimum heat.

VLSI placement usually involves two steps:

1. **global placement**: all cells that have to be placed are roughly spread out over the chip area ignoring overlapping. Emphasis is on minimising wire length, while spreading the cells over the available area.

2. **detailed placement**: In this step all cells are appointed a legal position. This includes removing all overlap and alignment with cell rows.

<table>
<thead>
<tr>
<th>Global placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Relax the placement region. Allow overlapping.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Detailed placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Put the cells in standard rows by preserving the global placement as much as possible.</td>
</tr>
</tbody>
</table>

Table 1.1: Two steps in placement
**Placement objects**

The flat design of today’s high performance processor chips requires a program for placing several million objects with respect to wiring and timing constraints. In this report, the objects to be placed will be called *cells* (other authors also speak of modules, components or circuits). We assume that all of the cells are *standard cells*. A standard cell is a small unit, which performs a simple logical function. It can be considered as a rectangle, with pins(terminals) on its boundaries. A standard-cell placement problem is a given set of standard cells, together with a given set of nets, which interconnect the cells. All cells have equal heights, while their width may vary. Figure 1.2 shows a local part of a standard-cell layout where the cells are placed in a number of rows.

**Placement definitions**

We will give some basic definitions related to placement here, and figure 1.3 illustrates these definitions.

- **Cell**: a circuit component to be placed on the chip area. In placement, the functionality of the component is ignored.

- **Pins**: a number of input and output connections of the cells. For simplicity, we assume that the pins of a cell are located in its center. Each cell has 2 or more pins and each pin is connected to a net. Each net connects 1 or more pins.

- **Net**: a set of pins that must be electrically connected and the conducting wire needed to achieve this connection.

- **Wire length**: The total wire length is the sum of the wire length for each of the nets. Wires on a chip are generally routed in a Manhattan style, containing only horizontal and vertical segments.

![Figure 1.3: Cells, pins and nets in a standard cell layout](image-url)
1.3 Cell spreading

A common objective function in performing the global placement is minimizing total wire length such that the minimum total wire length of cells is obtained. One of the widely used approaches is force-directed placement[4]. This approach uses a quadratic formulation for the wire length minimization. By trying to minimize wire length while spreading, groups of tightly connected cells will be spread out continually and contending with neighboring groups for space. In the last phase of this wire length minimization placer, some cells still overlap in some layout regions such that these crowded regions become clouds, these clouds are settled and should have to spread further to obtain an even distribution.

This last phase will take quite a few iterations, but the relative placement does not change so much. Such a placement is not adequate for creating legal placement as the amount of overlap among the cells is quite large. This is because the quadratic formulation for the wire length minimization ignores the requirement that cells should not overlap. To spread the overlapping cells, motivates us to seek an iterative process as a means of spreading out the cells before the detailed placement. This process will start with an initial placement and then modify it so that it is improved in some way while still preserving the essential nature (cell ordering, wirelength, etc) of the original placement which is the result from the computation of the global placement. And the work of this process will reduce the difficulties for the detailed placement.

Figure 1.4: The trajectory of placement design
Therefore, we want to add a new step into the cell placement procedure. The new step “cell spreading” may have various functions:

- Allow the global placer to be stopped in an earlier stage.
- Make sure the detailed placer can place all cells close to their given location.
- Reducing routing congestion by spreading the empty space equally over the area.

This new cell spreading step is added before the detailed placement, as shown in the following flow chart (see figure 1.4). Some cell spreading methods will be discussed in this report.

### 1.4 Outline

This section describes the organization of this thesis.

**Chapter 2** gives the mathematical notations and formulae considered during the course of this project. It Describe the placement model for testing, the testing model, the framework of the placement tool, and testing measurements.

**Chapter 3** introduces a simple pair-movement algorithm, afterwards in an experiments example, lists results we obtained on running the algorithms for different sets of values.

**Chapter 4** discusses the details of the algorithm with the electrostatic model, data structures used, and the principle behind the choice of data structures.

**Chapter 5** presents a diffusion-based algorithm which generates cell velocities from local density distribution and uses integral equations of velocities to directly compute the cell locations.

**Chapter 6** lists results we obtained by running the algorithms for different sets of values. We draw some conclusions on the controlling of algorithm parameters and compare the performance of the algorithms.

**Chapter 7** summarizes the cell spreading methods were done in this project. We list out future work that could be done as an extension of this project.
Chapter 2

Fundamental mathematical notations and the placement model

This chapter presents the basic mathematics notations and formulae which will be used in this thesis. The general initial situation of cell placement is described in Section 2.1. To spread the cells apart in the placement domain, one important definition “density” is discussed in Section 2.2. We define the goal of cell spreading and some typical situations following. Then we present a standard cell placement simulated model in Section 2.5. The placement problem will be formulated in a typical situation which is based on this simulated model. The details about the cell spreading tool are discussed in Section 2.6. We will talk about the framework of the cell spreading tool and the testing measurements for verifying the testing results.

2.1 Initial situation

![Figure 2.1: The initial situation of the placement](image-url)

Figure 2.1: The initial situation of the placement
We have a domain $\Omega \subset \mathbb{R}^2$ in which a set of $n$ rectangular cells $C_i \subset \Omega$ are placed. For each cell $i$, the width $w_i$ and height $h_i$ are given, as well as the initial location of the center, $(x_i^{(0)}, y_i^{(0)}) \in \mathbb{R}^2$. The cells will be approximately equi-spread over the domain. Initially, we will consider a rectangular domain $\Omega = [0, H] \times [0, W]$. Figure 2.1 describes the initial situation.

### 2.2 Notations

For convenience, we introduce some notations.

- **Area $A_\Gamma$**
  The area of a region $\Gamma \subset \mathbb{R}^2$ is denoted by $A_\Gamma$.

- **Coordinates of all cells**
  It is convenient to have a notation for the sequence of coordinates of all cells. We denote this with $X \in \mathbb{R}^{2 \times n}$, and $X_i \in X, X_i = (x_i, y_i)^T$ for $i = 1, 2, \ldots, n$.

  $$X = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{pmatrix}.$$  

- **The density function**
  Here we give two density functions:

  1. **Density of point $(x, y)$**
     The density function $\rho(x, y) : \Omega \to \mathbb{R}$ denotes the occupation factor. At point $(x, y)$, $\rho(x, y)$ is defined by

     $$\rho(x, y) = \sum_{i=1}^{n} \rho_i(x, y) \; , \tag{2.1}$$

     where

     $$\rho_i(x, y) = \begin{cases} 1 & \text{if cell } i \text{ covers } (x, y) \iff |x - x_i| < \frac{w_i}{2} \text{ and } |y - y_i| < \frac{h_i}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

  2. **Density of a region**
     For region $\Gamma$ of the placement domain $\Omega$, some cells are overlapping with the region $\Gamma$. The sum of all these overlapping area may be larger or smaller compared with the area of this region. We define the density of a region as,

     $$\rho_\Gamma = \sum \frac{A_{c_i \cap \Gamma}}{A_\Gamma} \; , \tag{2.2}$$

     where $A_{c_i \cap \Gamma}$ is the overlapping area of cell $i$ and region $\Gamma$. $A_\Gamma$ is the area of the region $\Gamma$. 

8
• **Average density**

The average density over a region $\Gamma \subset \Omega$ is denoted by $R(\Gamma)$ and is defined by

$$R(\Gamma) = \frac{1}{A_\Gamma} \int \int \rho(x, y) \, dx \, dy, \quad (2.3)$$

where, $\rho(x, y)$ is defined by (2.1).

We notice that actually the average density over a region equals the density of a region, $R(\Gamma) = \rho_{\Gamma}$. If we extend this formula to the whole domain, the average density over the domain follows that,

$$R(\Omega) = \sum_{i=1}^{n} w_i \times h_i W \times H. \quad (2.4)$$

• **Scaled density**

The scaled density of region $\Gamma$ is denoted by $\rho_{s\Gamma}$ and defined by

$$\rho_{s\Gamma} = \frac{\rho_{\Gamma}}{R(\Omega)}. \quad (2.5)$$

### 2.3 Goal

To spread out the overlapping cells to gain more even placement distribution, we add cell spreading as a new step in between global placement and detailed placement. The work of cell spreading reduces the number of overlapping cells while it does not disturb the properties of the original placement too much. Such properties contain the cell ordering and minimization of the wirelength.

In other words, our goal is to evenly spread the cells over the domain with minimal cell movement. Both the terms “evenly” and “movement” are not rigorously defined, but we will give possible definitions below.

**Goal 1 – Even distribution**

**Definition** : A placement of cells is said to be evenly distributed with parameter $\rho_M$\(^1\) and a set of windows\(^2\) $(W_i)_{i=1...m}$, if all of the following requirements are met:

1. All the cells are inside the domain, $C_i \subset \Omega$
2. For each window $W_i$, $\rho_{w_i} \leq \rho_M$

It can be easily shown that such a distribution does not exist if $\rho_M \leq R(\Omega)$.

\(^1\) $\rho_M$: the maximum density parameter

\(^2\) The chip area is divided into $m$ equal sized boxes, we call such boxes windows.
Goal 2 – Minimal movement

The initial placement $X^{(0)}$ is the result of a previous optimisation process. We want to preserve this optimality as much as possible. Again, it is not clear how to formulate this without considering the optimised properties itself. The general idea is to try to keep cells close to their surrounding cells while doing the final spreading. The simplest way of looking at this is to try to minimise the movement of cells,

$$\min_{X_i \in X} \sum_i \|X_i - X_i^{(0)}\|,$$  \hspace{1cm} (2.6)

for some appropriate norm $\| \cdot \|$.

Some examples of norms that might be useful are

- the 1-norm: $|X|_1 = \sum_i |x_i| + |y_i|$
- the 2-norm: $|X|_2 = (\sum_i |x_i|^2 + |y_i|^2)^{1/2}$
- the $\infty$-norm: $|X|_\infty = \max_i \max(|x_i|, |y_i|)$
- a combination of norms: $|X|_{2,1} = (\sum_i (|x_i| + |y_i|)^2)^{1/2}$

Where $X$ is in vector form: $X = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$.

Since we prefer to move a lot of cells a little bit instead of a few cells a lot, the 2-norm or the combination norm seems to be more suited for this problem.

2.4 Typical situations

In order to get a feeling for the numbers and sizes involved, we will discuss here the ranges of values for the parameters we are interested in.

$n$ First of all, the number of cells $n$. For this project, we use the values of $n$ in the range of $n = 1000$ to $n = 10,000$. However, real large designs can contain several millions of cells.

$R(\Omega)$ Currently, the overall cell density, also called utilization, can range up to $95\%$, $R(\Omega) \leq 0.95$. However, more is better, so it would be good to be able to handle higher densities.

$\rho_M$ Typically, the parameter $\rho_M$ of the “even distribution” requirement will be a few percent higher than the utilization. For problems with a utilization close to $100\%$, this increment will be less in order to keep $\rho_M$ less than $100\%$.

For problems with low utilization, we do not want to force the cells to spread over the whole domain, resulting in large cell movement. To prevent unnecessary cell movement, $\rho_M$ will always be set to at least $60\%$. 
The cell aspect ratio is mostly limited to \([4, 0.1]\) (i.e., \(4 \leq h_i/w_i \leq 0.1\)).

A reasonable choice of windows would be to use a grid of square windows with an area in the range of 25 to 100 times the average cell area.

### 2.5 Standard cell placement simulated model

In Chapter 1, we propose cell spreading as a new step between global placement and detailed placement. Then the initial state of the placement model should comprise most characteristics of the result of the global placement. The global placement places emphasis on the wire length minimization. At the last phase of the global placement, some cells cluster together in some regions, these crowded regions look like clouds when the placement problem contains large number of cells. Moreover, such placement model should be somewhat difficult to spread evenly. To verify all algorithms of this project experimentally, the testing model for each algorithm is same, thus it is easy to compare the results of these algorithms.

In this project, the number of cells we consider is in the range of \(n = 1000\) to \(n = 10,000\). And the coordinates of all cells we will trace instantly during the spreading process, then we treat the coordinates of all cells as a set \(X\) which is defined in section 2.2. \(X\) can be consider as an independent variable. Intuitively, we simulate the initial placement with normal random distribution. We say that \(X\) is a normal random variable with parameters \(\mu\) and \(\sigma^2\), and the probability density of \(X\) is given by

\[
\begin{align*}
  f(x_i) &= \frac{1}{\sigma \sqrt{2\pi}} e^{-\left(x_i - \mu\right)^2/(2\sigma^2)}, \quad 0 < x_i < 1 \\
  f(y_i) &= \frac{1}{\sigma \sqrt{2\pi}} e^{-\left(y_i - \mu\right)^2/(2\sigma^2)}, \quad 0 < y_i < 1
\end{align*}
\]

(2.7)

where \(X_i \in X\), \(X_i = (x_i, y_i)^T\) for \(i = 1 \ldots n\).

The density function is a bell-shaped curve that is symmetric around mean \(\mu\). The variance \(\sigma^2\) determines the range of most cells’ coordinates. For example, if we set \(\mu = 0.5, \sigma^2 = 0.25\), that means the mean value of the cell’s coordinate is \((0.5, 0.5)\), most cells are located within \([0.25, 0.75] \times [0.25, 0.75]\)

We set up a normal random distribution testing placement model for this project. The placement domain is set to be: \(\Omega = [0, 1] \times [0, 1]\). And for every standard cell, we set cells in square size, their width and height are the same, i.e., \(w_1 = h_1\) and \(w_2 = w_3 = \cdots = \text{width, } h_1 = h_2 = \cdots = \text{height}\). In real design, the height of most standard cell are same, but the width might be different. Here, we simplify the situation of our model. Figure 2.2 shows an initial placement state which is generated in Matlab.
2.6 Density of each window

We partition the placement domain into windows, and determine the density of each window. For the testing placement model, we set the domain: \( \Omega = [0, 1] \times [0, 1] \). Here, we put densities of all windows into a matrix, corresponding to the placement domain with \( N \times N \) windows (see figure 2.3).

\[
\rho_w = \begin{pmatrix}
\rho_{w(1,1)} & \rho_{w(1,2)} & \cdots & \rho_{w(1,N)} \\
\rho_{w(2,1)} & \rho_{w(2,2)} & \cdots & \rho_{w(2,N)} \\
\vdots & \vdots & \ddots & \vdots \\
\rho_{w(N,1)} & \rho_{w(N,2)} & \cdots & \rho_{w(N,N)}
\end{pmatrix}
\]

Figure 2.2: The initial placement of the standard cell placement simulated model

Figure 2.3: Partition of the placement domain for windows
from (2.2), the density of window \((i, j)\) is:

\[
\rho_w(i,j) = \frac{\sum A_{c_k \cap W(i,j)}}{A_w(i,j)}
\] (2.8)

for \(i = 1, 2, \ldots, N; \ j = 1, 2, \ldots, N.\)

where, \(A_{c_k \cap W(i,j)}\): the overlapping area of cell \(k\) and window \((i, j)\);
\(A_w(i,j)\): the window \((i, j)\)'s area. \(A_w(i,j) = \Delta^2.\)

Then from (2.9), the scaled density of window is:

\[
\rho_{sw(i,j)} = \frac{\rho_w(i,j)}{R(\Omega)}
\] (2.9)

2.7 The cell spreading tool

2.7.1 The framework of the cell spreading tool

Some cell spreading methods will be discussed in this thesis. Here, we talk about the framework of the cell spreading tool. In order to measure the efficiency of a cell spreading method and to trace the placement migration, such tool should be iterative to implement the cell spreading method. Figure 2.4 presents the whole process of the framework. All the components are explained in the following.

Figure 2.4: The framework of the cell spreading tool
**Initial allocation:** In the initial routine, all cells are placed randomly in the placement domain. We use the standard cell placement simulated model to generate the initial placement state. One cell’s coordinate is represented as a vector, which is convenient for computing: $X_i = (x_i, y_i)^T$.

**Compute cell movement:** For different cell spreading method, we use different objective function to compute the movement displacement $\Delta X$ of every cell. These functions could be algebra function to determine which cells are overlapping (as we see the algorithm in next chapter “A pair-movement algorithm”), or could be an analytical function to introduce a “force” which is simulated to spread the cells from high density regions to lower ones (see chapter 5).

**Move cells:** After the displacement $\Delta X$ is calculated, the new placement at the moment $n + 1$ is obtained by:

$$X^{n+1} = X^n + \alpha \Delta X^n \quad \alpha : \text{magnitude factor} \quad (2.10)$$

Generally, it is better to choose $\alpha$ between $(0, 1)$. For $\alpha \in (0, 0.5)$, the cells movement will be more smooth than the case when $\alpha \in (0.5, 1)$.

**Stopping criterion:** Also, we want to implement the cell spreading methods as iterative algorithms, we stop the iterations when

$$\max(\rho_{w(i,j)}) \leq \rho_M \quad \text{or} \quad \text{iteration} = \text{number} \quad (2.11)$$

$max(\rho_{w(i,j)})$ is the maximum density of window, $\rho_M$ is the maximum density parameter, $\text{number}$ is the maximum iteration number.

From the definition of “even distribution” (see section 2.3), we say if the result placement satisfies both requirements of even distribution, one of the goals of cells spreading is reached. The result placement might be the best placement expected as we should measure whether the movement of cells is minimal.

A variation of this stop criterion is

$$\max(\rho_{sw(i,j)}) \leq \frac{\rho_M}{R(\Omega)} = \rho_{sM} \quad \text{or} \quad \text{iteration} = \text{number} \quad (2.12)$$

$max(\rho_{sw(i,j)})$ is the maximum scaled density of window (see the definition in section 2.6). $\text{number}$ is the maximum iteration number.

(i) Use the scaled density:

$$\rho_{sw(i,j)} = \frac{\rho_{w(i,j)}}{R(\Omega)}, \quad R(\Omega) : \text{the average density over the domain}$$

the ideal state is: $\rho_{sw(i,j)} = 1$ for all windows, that is $\rho_{w(i,j)} = R(\Omega)$, all cells are evenly distributed in the domain.
(2) In case the process will not attain the first stop criterion, we set the maximum iteration number as an auxiliary condition.

Summary: *The iterative cell spreading tool (algorithm)*

The complete algorithm consists of three key elements:

1. Initialization: We assign all the cells’ initial locations;
2. Iteration loop: We iteratively apply placement transformation. We use the parameter $\alpha$ to control the desired speed of the algorithm. Each iteration makes the distribution more even and adapts the placement more to the placement area;
3. Stopping criterion: Using “stopping criterion” to determine whether the result placement is good enough as we expected to exit the iterative loop.

### 2.7.2 Testing measurements

We set up 2 testing measurements based on the 2 goals for cell spreading.

- **Even distribution**
  
  We define a set $I$:

  $$I = \{(i, j) \mid \rho_{sw(i, j)} > 1\}$$

  Each element $(i, j)$ represents window $(i, j)$ of which the scaled density is larger than 1, that means these windows with higher densities than the average density over the domain. We are interested in how evenly the cells are distributed. To measure this, we introduce an even distribution measurement $s$:

  $$s = \frac{\sum_{(i,j) \in I}(\rho_{sw(i,j)} - 1) \times A_{w(i,j)}}{W \times H} \quad (2.13)$$

  The above part of this division equals the total overcrowded regions’ area. For some windows whose densities are higher than the average density, we should put more effort to distribute the cells in these windows more apart. Then we compare this total overcrowded regions’ area with the whole placement area, if the result placement with equal densities for all windows, i.e., $\rho_{w(i,j)} = R(\Omega)$, we obtain $s = 0$.

- **The total movement distance**

  After each iteration, every cell is moved a distance compared with the initial location. This distance is calculated in Frobenius norm 3, for it is

  3 Frobenius norm: $\|X\|_F = (\sum_{i=1}^{n} \sum_{j=1}^{n} |X_{ij}|^2)^{1/2}$, $X$ is a $2 \times n$ matrix
easy to associate with the displacement of all cells:

\[ d = \| X^n - X^0 \|_F \]

- \( X^0 \): the initial placement
- \( X^n \): the placement at moment \( n \)

### 2.7.3 Iterative scheme of the cell spreading tool

According to the framework of the cell spreading tool and the testing measurements, the procedure of the iterative cell spreading tool is listed below. We will perform different cell spreading method in this procedure.

---

**stop criterion** = 1;
allocate the initial placement;
partition the placement domain into windows;
iter = 0; \( \% \) initialize iteration times
iter\_max = number; \( \% \) set the maximum iteration times
input \( \alpha; \) \( \% \) the magnitude factor

**while** stop criterion == 1
	compute cell movement: \( \Delta X^{iter} \) \( \% \) use different objective function
	\( X^{iter+1} = X^{iter} + \alpha \cdot \Delta X^{iter} \); \( \% \) move cells

calculate \( \rho_{sw} \); \( \% \) the scaled density of each window
	if \( max(\rho_{sw(i,j)}) \leq \rho_sM \) or iter = iter\_max
	\( \) stop criterion = 0;
end
iter = iter + 1;
calculate \( s, d \); \( \% \) calculate measurements

---

Table 2.1: Iterative scheme of the cell spreading tool

### 2.8 Conclusions

There are three elements we should take into account for dimension: the size of every cell, the movement distance of all cells and the size of the placement domain. The width and height of a cell \((w_i, h_i)\), the total movement \((d)\), and the rectangular domain length elements \((W, H)\) should be in the same dimension with the same length unit. Therefore, based on this point, the important definition \textit{density} can be given. We gave possible definitions and notations in sec. 2.3 and sec. 2.3.

The placement simulated model is set up by using the normal random distribution function. With this model, we can generate different testing model, for example, a placement with one crowded group in the center of the domain or with several crowded groups distributed in the domain.
To measure the efficiency of different cell spreading method, we introduce two measurements: even distribution measurement and the total movement distance with respect to the two goals of cell spreading. We calculate these 2 measurements after the new placement is allocated in every iteration.
Chapter 3

Pair-movement algorithm

We begin to spread cells over the placement domain with a simple method “Pair-movement algorithm”. The generic idea of this algorithm is discussed in section 3.1. Then we describe the detailed implementation of this method in Section 3.2. We design a movement function which is based on determining the overlapping parts of every two cells. The setup of the experiment used to gather the data sets is presented in Section 3.3.

3.1 Overview of the algorithm

The initial placement state of cell spreading is the output of the global placement. In section 1.3 we mentioned that the number of overlapping cells in this initial placement state is quite large. The task of cell spreading is to spread all cells evenly over the placement domain while keeping the movement minimal. For the next step, detailed placement will work out on removing all overlap and align cells in cell rows (see section 1.2). Then intuitively, we want the cell spreading algorithm to move the crowded cells apart and to make them free of overlaps. Consequently, the cell spreading method with this intention leads to a result placement that will help the detailed placer to finish placement task in little time.

Following the previous idea, the general purpose of this “pair-movement

![Figure 3.1: The movement for 2 overlapping cells](image-url)
“algorithm” is that we try to move the overlapping cells apart. Imagine there are two cells which are overlapping each other, we will move them directly and the work should be minimal (see figure 3.1). Then if we set one cell is fixed without changing its location, for the other cell, there are 2 movement directions: horizontal motion and vertical motion. If we move the cell horizontally to make these two cells overlapping free, the cost (movement distance) is smaller than that was taken by moving the cell vertically, thus we will choose horizontal direction for cell movement.

Applying this procedure to all cells, we will get a resulting placement where all cells are moved a bit, and stopped at the locations which are near the original ones. Such operation is to spread out cells locally.

3.2 Implementation details

We describe the main three steps of this algorithm:

Determine the overlapping cells We choose one cell as fixed cell, compare it’s location with all the other cells:

fixed cell : \((x_f, y_f)\),
cell i : \((x_i, y_i)\), \(i = 1, 2, \ldots, n; i \neq f\),
the distance : \(d_1 = x_i - x_f\),
\(d_2 = y_i - y_f\),
the overlapping condition : \(|d_1| < \text{width}\), \& \(|d_2| < \text{height}\).

If two cells satisfy the overlapping condition, we say both of them are overlapping each other.
Design the movement function

\[ \Delta X_i = \text{sign} \ast \text{overlap} , \]

where, \( \Delta X_i = (\Delta x_i, \Delta y_i)^T \).

If there’s an overlapping part between 2 cells, there are 2 directions for the movement (horizontally or vertically), the movement distance is determined by measuring the overlapping part in which direction the move cost is less:

\[
\begin{align*}
O_1 &= w - |d_1|, & O_2 &= h - |d_2|, \\
\text{if} & \quad O_1 \leq O_2, & \text{overlap} &= O_1, \\
\text{if} & \quad O_1 > O_2, & \text{overlap} &= O_2.
\end{align*}
\]

sign = +/− 1, it depends on the position between every pair of cells: the fixed cell and the cell \( i \). Then,

For \( O_1 \leq O_2 \), if \( d_1 > 0 \), sign = +1, otherwise, sign = −1;
For \( O_1 > O_2 \), if \( d_2 > 0 \), sign = +1, otherwise, sign = −1.

Allocate the new placement

Computing the new coordinate of cell \( i \):

\[ X_i = X_i + \alpha \cdot \Delta X_i , \quad \alpha : \text{magnitude factor} \, (3.1) \]

We will do the same steps as before for the next cell (cell \((i + 1)\)). Then, the new placement at moment \( n + 1 \) can be obtained by

\[ X^{n+1} = X^n + \alpha \cdot \Delta X^n \, (3.2) \]

In case some cells’ new coordinates are outside the placement domain, the boundary constraint is embed into the iterative algorithm to move these cells back into the domain.

### 3.3 Experimental results

This section shows the results of our experiments on the performance of the “pair-movement algorithm”. We use the testing placement model which is introduced in section 2.5.

Table 3.1 lists the given data of our testing model. In this table, the explanation of the notations are

<table>
<thead>
<tr>
<th>width, height</th>
<th>the width and height of the standard cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>the number of cells</td>
</tr>
<tr>
<td>( R(\Omega) )</td>
<td>the average density over domain</td>
</tr>
<tr>
<td>( N_w )</td>
<td>the number of windows</td>
</tr>
<tr>
<td>( \text{iter}_{\text{max}} )</td>
<td>the maximum iteration number</td>
</tr>
<tr>
<td>( \rho_{\text{SM}} )</td>
<td>the maximum scaled density parameter</td>
</tr>
</tbody>
</table>

20
Table 3.1: Parameters of the testing model in chapter 3

<table>
<thead>
<tr>
<th>width = height</th>
<th>n</th>
<th>( R(\Omega) )</th>
<th>( N_w )</th>
<th>( \text{iter}_{\text{max}} )</th>
<th>stop criterion: ( \rho_{sM} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>400</td>
<td>1.0</td>
<td>16</td>
<td>2000</td>
<td>1.30</td>
</tr>
</tbody>
</table>

Figure 3.3 shows the initial placement and the initial density distribution.

![Initial Placement and Density Distribution](image)

Figure 3.3: The initial placement and the initial density distribution (in chapter 3)

For the density distribution of the initial placement, we count the number of windows with different scaled density of each window. There are 12 windows of which the scaled density is within \((0, 1.1)\). Some windows are with large density \((\rho_{sw(i,j)} > 1.5)\). These windows with high densities are located in the center of the placement domain, as we see from the graph of the initial placement.

In the experiments, we set \( \rho_{sM} \) and \( \text{iter}_{\text{max}} \) as the stopping criterion of this algorithm. To measure how even the resulting placement is, we pay more attention to 2 measurements: the even distribution measurement \( s \) and the maximum scaled density of window \( \max(\rho_{sw(i,j)}) \) (see the definitions in section 2.6, 2.7). Meanwhile, the total movement distance is examined to measure whether the work results in the minimal movement. The results with different magnitude factor \( \alpha \) are compared in the following table.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>iterations</th>
<th>( s )</th>
<th>( \max(\rho_{sw(i,j)}) )</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>2000</td>
<td>0.094719</td>
<td>1.370947</td>
<td>1.650935</td>
</tr>
<tr>
<td>0.4</td>
<td>712</td>
<td>0.097834</td>
<td>1.299920</td>
<td>1.799795</td>
</tr>
<tr>
<td>0.6</td>
<td>36</td>
<td>0.111391</td>
<td>1.293470</td>
<td>1.976404</td>
</tr>
<tr>
<td>0.8</td>
<td>22</td>
<td>0.106889</td>
<td>1.297752</td>
<td>2.423884</td>
</tr>
<tr>
<td>1.0</td>
<td>2000</td>
<td>0.226925</td>
<td>2.091180</td>
<td>0.908878</td>
</tr>
</tbody>
</table>

Table 3.2: The results of the experiment example in chapter 3
When \( \alpha = 0.2 \), the program will stop at the maximum iteration times which we set as one of the stop criterion. The maximum scaled density doesn’t reach to \( \rho_{sM} (= 1.3) \). When \( \alpha = 0.6 \), the maximum scaled density of window \( \max(\rho_{sw(i,j)}) \) is the least in table 3.2, and the total movement distance is smaller than the case \( (\alpha = 0.8) \). It is easy to see that \( \alpha = 1.0 \) is not a good choice. The maximum scaled density of window are higher than 2.0 when choosing \( \alpha = 1.0 \).

Our goal is to get an even distribution. Then for this example, the ideal scaled density of each window equals 1.0, and the even distribution measurement \( s = 0 \). From the results in the figure (3.4 – 3.6), when \( \alpha = 0.2 \) the curves are more smooth, we say in this case the movement of the cell spreading is more stable. For \( \alpha = 1.0 \), the curves attain a certain data in a few iterations and these data are far from the ideal results we expect. We notice that the even distribution measurement \( s \) decreases in the beginning few iterations and then stagnates in the rest (see figure 3.5, when \( \alpha = 0.2, \alpha = 0.4, \alpha = 1.0 \)). Because this algorithm is based on the overlapping part of every pair cells, it means if there’s no overlap in the placement, the cells will not be moved again. This can also be seen from the curve of the total movement distance in each iteration (figure 3.6).

![Figure 3.4: The maximum scaled density of window with different \( \alpha \) (in chapter 3)](image)

In this example, we set the average density equal to 1.0, that means for the ideal resulting placement, all cells are moved apart from each other and fully occupy the placement domain. In figure 3.7, the cells are spread least for the case \( \alpha = 1.0 \), but the resulting placement is not easy to distinguish from the
Another apparent comparison among these results is the scaled density distribution (see figure 3.8). If the scaled densities of all windows are around 1.0, then the resulting placement is good. The scaled density distribution of $\alpha = 0.8$ is better than for other cases.

From the above assessment, to obtain a fast cell spreading, we could set the magnitude factor $\alpha$ to be a large number. For this experiment example with $R(\Omega) = 1.0$, it is better to choose $\alpha < 1.0$. Thus we set the magnitude factor to control the speed of the placement migration. To keep cells close to their surrounding cells, we hope the movement distance is minimal and the movement is stable, this can be achieved by setting the magnitude factor a small number. For this experiment example, $\alpha = 0.6$ is the best choice.

For the case when $R(\Omega) < 1.0$

If the testing placement model with the average density $R(\Omega) < 1.0$, namely the placement area will not be fully occupied by all the cells, there’s more free space for moving the overlapping cells. For $0.6 < R(\Omega) < 1.0$, the behavior of this algorithm is similar with the previous example ($R(\Omega) = 1.0$). We also can set different magnitude factor $\alpha$ to control the cell’s movement speed. For the utilization $R(\Omega) \leq 0.6$, applying this cell spreading method, there will be no overlap left in the resulting placement. But it is not economic with such low utilization for practical placement. So we will not discuss any cases with
$R(\Omega) < 0.6$, and $\rho_{sM}$ will always be set to at least 0.6.

### 3.4 Conclusions

This chapter has illustrated a “Pair-movement algorithm” to spread cells by introducing a movement distance function. The distance of cells’ movement is determined by the overlapping part between every pair cells, as this algorithm is named.

Based on the framework of cell spreading tool (see section 2.6), the scheme of this algorithm has been shown in section 3.2. Applying this scheme, some experiments are done to test how does this algorithm work on cell spreading.

Finally, we remark that the cells spreading speed can be accelerated by setting large magnitude factor. And we can make the cells movement stable by setting somewhat small magnitude factor. For different testing case, the optimal choice of the magnitude factor will be different. For the experiment example in this chapter, it’s better to choose this magnitude factor around 0.6.
Figure 3.7: The result placement with different magnitude factor (in chapter 3)
Figure 3.8: The movement distance with different magnitude factor $\alpha$ (in chapter 3)
Chapter 4

Force-based algorithm with an electrostatic model

A force method for cell spreading is presented in this chapter. We apply this force with an electrostatic model to reduce cell overlaps and to distribute cells over the placement area. In Section 4.2, to obtain the new placement, we relate the displacement with the density state through deriving the Poisson equation for the electric potential. And in Section 4.3, we give the numerical solution to solve the related boundary value problem (BVP) and to get the gradient of the potential. Also, an experiment is implemented to verify this method in the last part of this chapter.

4.1 The force for cell spreading

For the global placement, one of the widely used approaches is forced-directed method. This method simulates the wire-length dependent forces as spring forces where a quadratic objective function is used to model wire length. Eisenmann[5] introduces an additional force based on cell distribution to push cells away from dense region during the global placement. Therefore, that might be interesting for us to seek a force working on cells to move the overlapping cells apart. In nature, the electric force has a common property that the electric force is a repelling force between two charges of the same parity.

Objects that exert electric forces are said to have charge. Charge is the source of electrical force. There are two kinds of electrical charges, positive and negative. Same charges (+ and +, or - and -) repel and opposite charges (+ and -) attract each other. Of course, here we will simulate cells as same charged particles among which there exist repelling force. Now, we formulate the electrostatic force in 2-dimension:

\[
\vec{F} = k \sum_{i=1}^{N} \frac{q_{0} \cdot q_{i} (\vec{r}_{0} - \vec{r}_{i})}{|\vec{r}_{0} - \vec{r}_{i}|^2},
\]

(Coulomb’s Law) \hspace{1cm} (4.1)
where, $q_0, q_i$ are the charge of the particle 0 and particle $i$.
$r_0, r_i$ are the positions of the particle 0 and particle $i$ respectively.
$k$ is a constant, $k = 1/4\pi\varepsilon_0$.

The magnitude of force that a particle exerts on another particle is directly proportional to the product of their charges and inversely proportional to the square of the distance between them. The direction of the force is on the line from one particle to the other.

These physical properties of electric force illuminate us that the placement can be simulated as an electric field:

- Every cell is considered as a positive charged particle.
- Between every pair of cells, there lies electric force. The force between two cells should be same, and the force results in these two cells repel each other. If some cells overlap, the force will be large enough to move them apart.
- The force direction between each pair cells is on the line from one cell to the other.

Moreover, the relation between the electric field and the electric force is
\[
\vec{E} = \frac{\vec{F}}{q} . \tag{4.2}
\]
And there exists a scalar function $\Phi$ (electric potential) for $E$ such that,
\[
\vec{E} = -\nabla \Phi , \tag{4.3}
\]
\[
\Rightarrow \vec{F} = -q\nabla \Phi . \tag{4.4}
\]

### 4.2 The mathematical model

We will derive the mathematical model for cell spreading with the electric force in this section. The process of deriving is to find the relation for the electric force and the density field.

From the Maxwell's equation and with equation(4.3):
\[
\nabla \cdot (\varepsilon E) = \rho ,
\]
\[
\Rightarrow \nabla \cdot (-\varepsilon \nabla \Phi) = \rho ,
\]
\[
\Rightarrow \varepsilon \nabla^2 \Phi + \nabla \Phi \cdot \nabla \varepsilon = -\rho .
\]
If \( \varepsilon \) is constant related to the media in this electric field, then we have:

\[
\nabla^2 \Phi = -\frac{\rho}{\varepsilon} .
\]

(4.5)

For the purpose of simplification, take \( \varepsilon = 1 \), rewrite equation (4.5):

\[
\nabla^2 \Phi = -\rho . \quad \text{(poisson equation)}
\]

(4.6)

We know that for any objects, a force working on it will give an acceleration,

\[
\vec{F} = m\vec{a} , \quad \text{(Newton’s second law)}
\]

\[
\Rightarrow \quad \vec{a} = \frac{\vec{F}}{m} = -\frac{q}{m} \nabla \Phi , \quad m : \text{ mass of the object.}
\]

This force works on one cell, resulting in a displacement for this cell

\[
\Delta X_i = \vec{a}\Delta t = \vec{a}\Delta t \cdot \Delta t = \vec{a}(\Delta t)^2
\]

\[
= -\frac{q}{m} \nabla \Phi (\Delta t)^2 .
\]

Set \( \Delta t = 1 \) (unit time). Also for simplification, we set \( q/m = 1 \). Then the new coordinate of one cell at moment \( n + 1 \) is obtained:

\[
X_i^{n+1} = X_i^n + \Delta X_i^n = X_i^n - \nabla \Phi , \quad \text{(4.7)}
\]

and equation (4.7) can be written in the following form:

\[
X^{n+1} = X^n + \alpha \cdot \Delta X^n , \quad \text{(4.8)}
\]

where,

\[
X = \begin{pmatrix}
  x_1 & x_2 & \ldots & x_n \\
y_1 & y_2 & \ldots & y_n
\end{pmatrix}.
\]

\( \alpha \) is the magnitude factor which is used to control the speed for the movement. For cell \( i \), the location is \((x_i, y_i)\) represented by the center coordinate of the cell \( i \).

From (4.4) and (4.6), here, we list them together

\[
\vec{F} = -q\nabla \Phi . \quad \text{(4.9)}
\]

\[
-\nabla^2 \Phi = \rho . \quad \text{(4.10)}
\]

If we have an initial placement, with the initial density \( \rho_0 \), the potential \( \Phi \) can be computed by equation (4.10). After calculating the gradient of potential \( \Phi \) by (4.9), from equation (4.8), every cell’s new coordinate can be achieved.

Consider the boundary constraint: all cells should be moved within the placement domain, it means when an arbitrary cell is moved to and touch the border of the domain, the force acted on this cell with the normal direction to
the border should be disappear. Then, the boundary conditions can be derived as:

\[ \mathbf{F}_\perp = 0 \quad \text{, } \perp: \text{ the normal direction to the boundary,} \]
\[ \Rightarrow \nabla \Phi \cdot \mathbf{n} = 0 \quad \text{, } \mathbf{n}: \text{ the unit vector with normal direction to the boundary.} \]

The force is computed as a function of the potential \( \Phi \), the potential \( \Phi \) is the connection between the force and the placement density field \( \rho \). In this approach, finding a placement is transformed into determining the cell force.

### 4.3 The numerical solution

Thus, in order to get the new placement, first we should solve the following BVP:

\[
\begin{aligned}
-\nabla^2 \Phi &= \rho, \quad (x, y) \in \Omega, \\
\nabla \Phi \cdot \mathbf{n} &= 0, \quad (x, y) \text{ at } \partial \Omega.
\end{aligned}
\]  

(4.11)

#### 4.3.1 Discretization

This section is to derive a discrete form of the BVP (4.11). We will use the Finite Difference method to formulate this discrete form.

Let us consider the Poisson equation for the domain \( \Omega = [0, 1]^2 \) with the Neumann boundary condition. We discretize the square \( \Omega \) into a rectangular mesh with \( N + 1 \) grid points in each dimension (see figure 4.1). Let \( \Phi_{i,j} \) denote the value of \( \Phi \) at the grid point \((i - 1)\Delta, (j - 1)\Delta\), where, \( \Delta = 1/N; \quad i = 1, 2, \ldots, N, (N + 1); \quad j = 1, 2, \ldots, N, (N + 1) \)

\[ \Delta = 1/N \]

Figure 4.1: The grids of the domain
Let $\rho_{i,j}$ denote the element of vector $\rho$ that corresponds to the grid point $((i-1)\Delta, (j-1)\Delta)$. We discretize the domain $\Omega$ into rectangular grids (see figure 4.2). We say grid cell $G(i, j)$, that grid point $((i-1)\Delta, (j-1)\Delta)$ in this grid. In the domain, the area of grid is $\Delta^2 = 1/N^2$. For the grid points on the boundary, the area of the grid cell equals to $\frac{1}{2} \Delta^2$, except for the area of grids which are located on the four corners of the square domain, the area of these four corner grids equals $\frac{1}{4} \Delta^2$.

![Figure 4.2: The meshes for calculating density](image)

And we calculate the density $\rho_{i,j}$ by computing the sum of the area of the cells which overlap with the grid cell $G(i, j)$, that is,

$$\rho_{i,j} = \sum \frac{A_{c \cap G(i,j)}}{A_{G(i,j)}},$$  \hspace{1cm} (4.12)

where, $A_{c \cap G(i,j)}$: the overlapping area of cell $i$ and grid cell $G(i, j)$; $A_{G(i,j)}$: the area of grid cell $(i, j)$.

Let $A$ be the $(N+1)^2 \times (N+1)^2$ coefficient matrix obtained by discretizing the Poisson equation. $A\Phi$ is then the $(N+1)^2 \times 1$ vector obtained by multiplying $A$ with the vector $\Phi$. From the BVP(4.11), we have

$$-\nabla^2 \Phi = \rho,$$

$$\Rightarrow - \left( \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} \right) = \rho.$$  \hspace{1cm} (4.13)

With the Finite Central Difference (FCD), we obtain the discrete form of the equation at point $(i\Delta, j\Delta)$:

$$\frac{\partial^2 \Phi}{\partial x^2} = \frac{\Phi_{i-1,j} - 2\Phi_{i,j} + \Phi_{i+1,j}}{\Delta^2},$$

$$\frac{\partial^2 \Phi}{\partial y^2} = \frac{\Phi_{i,j-1} - 2\Phi_{i,j} + \Phi_{i,j+1}}{\Delta^2}.$$
Combining these two equations, for an internal point, the left hand side of the Poisson is equal to the discrete Laplacian:

\[-\frac{1}{\Delta^2} (\Phi_{i-1,j} + \Phi_{i+1,j} - 4\Phi_{i,j} + \Phi_{i,j+1} + \Phi_{i,j-1}) = \rho_{i,j} \quad . \tag{4.14}\]

For the boundary condition,

\[\nabla \Phi \cdot \vec{n} = 0 \quad ,\]

with the Forward Finite Difference and the Backward Finite Difference,

\[\frac{\Phi_{i+1,j} - \Phi_{i,j}}{\Delta} = 0, \quad \frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta} = 0 \quad , \quad \text{(Forward)}\]

\[\frac{\Phi_{i-1,j} - \Phi_{i,j}}{\Delta} = 0, \quad \frac{\Phi_{i,j-1} - \Phi_{i,j}}{\Delta} = 0 \quad . \quad \text{(Backward)}\]

We have four different boundary condition formulae corresponding to the four borders of the square domain. Here, we use the pseudo grids on the borders (see figure 4.1):

On the left boundary:

\[x = 0 (i = 1), \quad y = (j - 1)\Delta, \quad \nabla_x \Phi = 0 \Rightarrow \Phi_{i,j} = \Phi_{0,j}, \quad j = 1, 2, \ldots, N + 1. \quad (4.15)\]

On the right boundary:

\[x = 1 (i = N + 1), \quad y = (j - 1)\Delta, \quad \nabla_x \Phi = 0 \Rightarrow \Phi_{(N+2),j} = \Phi_{(N+1),j}, \quad j = 1, 2, \ldots, N + 1. \quad (4.16)\]

On the bottom boundary:

\[x = (i - 1)\Delta, \quad y = 0 (j = 1), \quad \nabla_y \Phi = 0 \Rightarrow \Phi_{i,1} = \Phi_{i,0}, \quad i = 1, 2, \ldots, N + 1. \quad (4.17)\]

On the top boundary:

\[x = (i - 1)\Delta, \quad y = 1 (j = N + 1), \quad \nabla_y \Phi = 0 \Rightarrow \Phi_{i,N+2} = \Phi_{i,N+1}, \quad i = 1, 2, \ldots, N + 1. \quad (4.18)\]

In the domain \(\Omega\), we discretize first equation (Poisson equation) of (4.11) by using Finite Central Difference (FCD). On the boundary, the condition was discretized by Forward Difference (FD) and Backward Difference (BD), with (4.14) and (4.15)-(4.18), then BVP (4.11) reduce to a linear system:

\[A\Phi = \rho \quad , \quad (4.19)\]

and the following coefficient matrix for the discretized Poisson equation:

\[A = \begin{pmatrix}
J & -I & \cdots \\
-I & K & -I \\
\vdots & \ddots & \ddots \\
-I & K & -I \\
\cdots & \cdots & \cdots \\
-I & K & -I \\
\cdots & \cdots & \cdots \\
-1 & \cdots & \cdots \\
-1 & \cdots & \cdots \\
\end{pmatrix} \quad , \quad (4.20)\]
where $I$ is the $(N + 1) \times (N + 1)$ identity matrix, and

$$J = \begin{pmatrix} 2 & -1 & \ldots & & -1 \\ -1 & 3 & -1 & & \\ & -1 & 3 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 3 & -1 \\ & & & & \ddots & \ddots & \ddots \\ & & & & & -1 & 3 & -1 \\ & & & & & & \ddots & \ddots & \ddots \\ & & & & & & & \ddots & \ddots & \ddots \\ & & & & \end{pmatrix}, \quad (4.21)$$

$$K = \begin{pmatrix} 3 & -1 & \ldots & & -1 \\ -1 & 4 & -1 & & \\ & -1 & 4 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 4 & -1 \\ & & & & \ddots & \ddots & \ddots \\ & & & & & -1 & 4 & -1 \\ & & & & & & \ddots & \ddots & \ddots \\ & & & & \end{pmatrix}. \quad (4.22)$$

The properties of the coefficient matrix $A$:

- **Symmetric semi-positive definite**: all the eigenvalues of matrix $A$ are non-negative values.
- **Singular**: the rank of matrix $A$ is $(N + 1)^2 - 1$
- **Sparse matrix**

The sparsity structure of $A$ is shown in figure 4.3.

![Figure 4.3: The sparsity structure of the matrix A](image-url)
4.3.2 The singular linear system

We now consider the singular linear system $Ax = b$ which consists only one zero eigenvector. For this singular system, it have non-unique solutions. We would like to use LU-decomposition to solve the system in a direct way. Since matrix $A$ is singular, in LU decomposition, the U part will have a zero at the last diagonal element. This problem can be solved by simply replacing this zero pivot by the value 1 and removing the singular direction from the residual and the solution [6].

With the LU-decomposition, we have

$$PA = LU,$$

where

- $P$ – a permutation matrix
- $L$ – lower triangular matrix with unit diagonal
- $U$ – upper triangular matrix with a non-zero diagonal except for one zero at the last diagonal position

By replacing the zero by one, we obtain a regularised matrix $\tilde{A}$,

$$\tilde{A} = P^{-1}L(U + R) = A + P^{-1}R \quad ,$$

(4.23)

where $R$ is the zero matrix with a 1 at the last diagonal position.

$$R = e_n e_n^T = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}.$$  

And we change $b$ into a new vector which does not contain a component in the zero eigenvector direction.

$$b = Sb \quad .$$

(4.24)

Since the matrix $A$ is symmetric and the zero eigenvector $v_1$ is real and orthogonal to the other eigenvectors, then $S$ is an orthogonal projection:

$$S = I - v_1 v_1^T / \|v_1\|^2$$

(4.25)

And we also should make sure that the solution $x$ does not contain a component in the zero eigenvector direction. This can be achieved by applying the orthogonal projection again. Then we obtain the solution,

$$x = S \tilde{A}^{-1}Sb,$$

$$\Rightarrow x = S(U + R)^{-1}L^{-1}PSb \quad .$$

(4.26)
4.3.3 The gradient of the potential

After calculating the potential by solving the poisson problem, in order to get the new placement, we should find the gradient of the potential, that is to find the force for cell spreading. Using Finite Central Difference (FCD) again, we can obtain the gradient of potential of the point which is located between two grid points. For the gradient of potential of every cell, as the grid cell is small and to make sure that the gradient of potential is continuous over the placement domain, we consider the gradient of potential of one cell is bi-linear interpolate with the four gradients around this cell. And we just take account into the gradient of potential which is acted in the center of every cell. Figure 4.4 explains this.

With FCD, we calculate the gradient of potential,

\[
\frac{\partial \Phi}{\partial x}_{i+\frac{1}{2},j-1} = \frac{\Phi_{i,j-1} - \Phi_{i,j}}{\Delta}, \quad \frac{\partial \Phi}{\partial x}_{i+\frac{1}{2},j} = \frac{\Phi_{i+1,j} - \Phi_{i,j}}{\Delta}, \\
\frac{\partial \Phi}{\partial y}_{i+1,j+\frac{1}{2}} = \frac{\Phi_{i+1,j+1} - \Phi_{i+1,j}}{\Delta}, \quad \frac{\partial \Phi}{\partial y}_{i,j+\frac{1}{2}} = \frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta},
\]

where, \( \Delta = 1/N \).

To compute the gradient of potential of every cell, we should determine in which grid with the four gradients of potential the cell is located, that grid is different for calculating \( \frac{\partial \Phi}{\partial x} |_{(x,y)} \) and \( \frac{\partial \Phi}{\partial y} |_{(x,y)} \). Here, the location of the cell is represented by the coordinate in the center of it. For \( \frac{\partial \Phi}{\partial x} |_{(x,y)} \) (the gradient of potential in x direction), it is determined by \( \frac{\partial \Phi}{\partial x} \) at four points: \((i - \frac{1}{2}, j), (i + \frac{1}{2}, j), (i, j + \frac{1}{2}), (i + 1, j + \frac{1}{2})\). For \( \frac{\partial \Phi}{\partial y} |_{(x,y)} \) (the gradient of potential in y direction), it is determined by \( \frac{\partial \Phi}{\partial y} \) at four points: \((i + \frac{1}{2}, j - \frac{1}{2}), (i + \frac{1}{2}, j + \frac{1}{2}), (i, j + \frac{1}{2}), (i - \frac{1}{2}, j + \frac{1}{2})\).
\( \frac{1}{2}, j \), \((i - \frac{1}{2}, j - 1)\), \((i + \frac{1}{2}, j - 1)\). Similarly, for \( \frac{\partial \Phi}{\partial y}(x, y) \) (the gradient of potential in y direction), it is determined by \( \frac{\partial \Phi}{\partial y} \) at four points: \((i, j - \frac{1}{2})\), \((i + 1, j - \frac{1}{2})\), \((i, j + \frac{1}{2})\), \((i + 1, j + \frac{1}{2})\).

With the linear interpolation, we know for the point \( x \) (in figure 4.5), the value of function \( g(x) \) is:

\[
g(x) = \frac{x_1 - x}{x_1 - x_0} g_0 + \frac{x - x_0}{x_1 - x_0} g_1,
\]

(4.27)
with \( g(x_0) = g_0 \), \( g(x_1) = g_1 \)

![Figure 4.5: The linear interpolation](image)

Applying linear interpolation twice, first we calculate \( \frac{\partial \Phi}{\partial x} \) at \( A_1, A_2 \) and \( \frac{\partial \Phi}{\partial y} \) at \( B_1, B_2 \) (see figure 4.4). Then using linear interpolation again, we obtain the gradient of potential of every cell,

\[
\nabla \Phi = \begin{pmatrix}
\frac{\partial \Phi}{\partial x} \\
\frac{\partial \Phi}{\partial y}
\end{pmatrix}.
\]

(4.28)

For the gradient of the potential at the boundary, we should be aware of \( \nabla \Phi \cdot \vec{n} = 0 \). That means \( \frac{\partial \Phi}{\partial x} = 0 \) (at the horizontal boundary) and \( \frac{\partial \Phi}{\partial y} = 0 \) (at the vertical boundary). Then by adding the presudo points outside the domain, for some cells near the boundary, the gradient of potential of these cells can be determined by the linear interpolation method mentioned above.

Another way to think about making the gradient of potential continuous over the placement domain is to use the lowest-order Raviart-Thomas(RT) elements [7]. The Raviart-Thomas finite element space is defined as

\[
RT(\Omega_F) = \{ v \in H(div, \Omega_F) : v \mid_K = (a + bx, c + by), a, b, c \in \mathbb{R} \}
\]

\( \forall \) element \( K \in T_F, v \cdot n \) is cont. and const. on the element sides,

where \( H(div, \Omega_F) = \{ v \in L_2(\Omega) : \nabla \cdot v \in L_2(\Omega) \} \)

So with Raviart-Thomas elements, we just can guarantee the gradient of potential is continuous within every grid cell while on the border of every grid cell, it is not true.
4.4 Examples of application

4.4.1 The cell’s movement under the force

We set up the mathematic model based on the electrostatical model in this chapter. The electric force is the connection between the displacement of cells and the density state. Instead of calculating the force, we computing the gradient of potential by solving the related BVP (in section 4.3). And we use the reflective boundary condition, that is when a cell on the border of the domain there’s no force in the normal direction with the boundary for this cell. If there are 2 charged particles (cells) with the same electric charge quantity, one is just on the boundary, the other is in the placement domain, then there must exist a presudo charged particle at the symmetric place outside the domain. Figure 4.6 shows this state.

Figure 4.6: The state of the cell on the boundary

Because of this reflective boundary condition and with the repelling force between every 2 cells, all the cells will be moved and finally in a equilibrium state, that’s the resulting placement we expect. Figure 4.7 lists the final situation of the placement of only a few cells placement.

Figure 4.7: The result placement: equilibrium state
4.4.2 Experiments

We choose a testing case with 2500 cells to implement this force based method. In this chapter, we use the convergence measurement as the program stop criterion. Table 6.1 lists the given data of our testing model. The explanations of these notations are

- $n$: the number of cells
- $R(Ω)$: the average density over domain
- $N_w$: the number of windows
- $N$: the number of grid cells on each dimension
- $\text{iter}_{\text{max}}$: the maximum iteration number
- $ρ_{sM}$: the maximum scaled density parameter

<table>
<thead>
<tr>
<th>width = height</th>
<th>$n$</th>
<th>$R(Ω)$</th>
<th>$N_w$</th>
<th>$N$</th>
<th>$\text{iter}_{\text{max}}$</th>
<th>stop criterion: $ρ_{sM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02</td>
<td>2500</td>
<td>1.0</td>
<td>25</td>
<td>100</td>
<td>2000</td>
<td>1.006</td>
</tr>
</tbody>
</table>

Table 4.1: Parameters of the testing model in chapter 4

Figure 4.8 shows the initial placement and the initial density distribution and the initial potential $Φ$.

![Figure 4.8: The initial placement and the initial potential distribution (in chapter 4)](image)

For this testing example, we choose the number of grids in each dimension $N = \frac{1}{2 \text{width}} = 1/0.01 = 100$. This number is proved a good choice through experiments. And we set the stop criterion of the algorithm as: the maximum scaled density $ρ_{sM} = 1.006$ or iteration number equals $\text{iter}_{\text{max}}$. Similar to the experiments of “pair-movement algorithm” in chapter 3, here several groups of
result graphs are compared in the following.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>iteration</th>
<th>$\max(\rho_{s,w}(i,j))$</th>
<th>$s$</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>70</td>
<td>1.005932</td>
<td>0.001379</td>
<td>5.305682</td>
</tr>
<tr>
<td>0.4</td>
<td>49</td>
<td>1.005852</td>
<td>0.001372</td>
<td>5.305205</td>
</tr>
<tr>
<td>0.6</td>
<td>25</td>
<td>1.005921</td>
<td>0.001382</td>
<td>5.307793</td>
</tr>
<tr>
<td>0.8</td>
<td>21</td>
<td>1.005811</td>
<td>0.001254</td>
<td>5.309093</td>
</tr>
<tr>
<td>1.0</td>
<td>16</td>
<td>1.005929</td>
<td>0.001640</td>
<td>5.319347</td>
</tr>
</tbody>
</table>

Table 4.2: The results of the experiment example in chapter 4

Implementing this method, the program stops when $\max(\rho_{s,w}(i,j)) \leq \rho_sM$.

From the results in Table 4.2, the maximum scaled density of each window and the total movement distance for different $\alpha$ are close, except that the iteration numbers with different $\alpha$ are different. And we can’t figure out the difference from the resulting placement graphs in Figure 4.12 by the naked eye. The cells are spread fully in the placement domain such that the domain is totally occupied. Also for the resulting scaled density distribution, the result density is around 1.0. Compared with the ideal scaled density ($\rho_s = 1$), these results are much better than for the initial density distribution.

From Figures 4.9, 4.10, 4.11, we see that the algorithm is unstable when $\alpha = 1.0$, and it converges the most fast to the stop criterion. Compare the results of two cases ($\alpha = 0.6$ and $\alpha = 0.8$), the algorithm reaches the stop criterion in few iterations for both cases; In figure 4.9, when $\alpha = 0.8$, the maximum scaled density of window decreases more faster than that of the case $\alpha = 0.6$; but the total movement distance of the case $\alpha = 0.8$ is larger than that of $\alpha = 0.6$.

If the density distribution is even in the placement domain, for this experiment example, the ideal result potential is equal to zero. Comparing the result potential distribution with the initial one, we notice that the scale of the result potential is much less than the initial one. When $\alpha = 1.0$ and $\alpha = 0.8$, the scale is $\times 10^{-4}$, others are $\times 10^{-5}$ (see figure 4.13). The next example will show why the magnitude factor should be $< 1.0$.

The experiment with $\alpha = 1.5$

Many cells are stacked around the square boundary in the result placement graph (see figure 4.12), so the potential is high around the boundary and goes down to the center of the placement area. The program converges very fast but unstable, as we see from the results in figure 4.14. The cells’ movement is in an oscillating way. We let the program stop at $\text{iter}_{max} = 2000$, it didn’t attain the stopping criterion $\rho_sM \leq 1.006$ we imposed. And the result potential distribution is in scale $\times 10^{-3}$, much larger than the other cases.
Figure 4.9: The maximum scaled density of window in each iteration (in chapter 4)

4.5 Conclusions

The spreading technique described in this chapter models the density map as an electric field whereby every region of the density map has some attraction or repulsion to every cell. The mathematical model of this method comprises a boundary value problem and the cell’s displacement formula. For the boundary value problem (BVP), we solve the Poisson equation with reflective boundary conditions. We discretize this BVP with the finite difference scheme, which results in a singular linear system.

For solving the singular system, we propose to use an LU-decomposition in a direct way by taking care of the solution without containing the component in the zero eigenvector direction.

Some applications of this algorithm are listed in the last section of this chapter. The experiment on the cell spreading problem has demonstrated very significant results on even distribution and the program runtime.
Figure 4.10: The even distribution measurement in each iteration (in chapter 4)

Figure 4.11: The movement distance in each iteration (in chapter 4)
Figure 4.12: The result placement with different magnitude factor (in chapter 4)
Figure 4.13: The result potential distribution
(a) The maximum scaled density of window

(b) The even distribution measurement

(c) The movement distance

Figure 4.14: The experiment with $\alpha = 1.5$ (in chapter 4)
Chapter 5

Diffusion-based placement algorithm

We present a diffusion-based method that has the advantage of smooth spreading which preserves the integrity of the original placement. This algorithm takes advantage of a discrete approximation to a closed form solution of the continuous diffusion problem formulation. Section 5.1 describes the formulation of the diffusion-based placement process. Section 5.2 gives the numerical method to simulate the diffusion process. The experimental results are shown in section 5.4, followed by the conclusions in section 5.5.

5.1 Cell spreading and diffusion process

Diffusion is a well-understood process that moves a physical particles (such as air molecules) from a state with non-zero potential energy to a state of equilibrium. The particles bounce back and forth between collisions and these motions due to gradients in concentration.

The placement problem for cell spreading can be described as: Given an existing placement \((x_i, y_i)\) for cell \(i\), how to gradually move cells to obtain a new placement \((x_i^*, y_i^*)\) such that all cell are evenly distributed over the placement domain. This cell movement process is similar to the diffusion process, which moves material from high concentration area to less concentrated area. Naturally, we can formulate the cell spreading process under the diffusion law, which is given in next section. In [8], a placement method also based on diffusion process is introduced for the placement legalization (detailed placement). Here, we apply the diffusion process for cell spreading.
5.2 Equations of the diffusion process

We simulate the cell’s movement as a diffusion process. Materials from highly concentrated areas would flow into less concentrated areas. Diffusion is driven by the concentration gradient, which is the slope and steepness of the concentration difference at a given point. And the increase in concentration in a cross section of unit area is simply the difference of the material flow into the cross section and the material flow out of it. The final equilibrium of diffusion is an equal concentration distribution.

From Fick’s First law of diffusion\(^9\), the diffusion flux is defined by
\[
\mathcal{F} = -D \nabla \rho ,
\]  
(5.1)
is proportional to the gradient of the fluid variable \(\rho(x, y, t)\). where, \(\rho\) is the material concentration, in our case, that is the cells’ density at point \((x, y)\); \(D\) is the diffusivity which determines the speed of diffusion, and is assumed constant in space and time.

The sign of the flux means large values of \(\rho\) tend to decrease as regions with lower values of \(\rho\) fill in. When \(\triangle t \rightarrow 0\), we consider the cells are in the steady state diffusion, i.e., when the concentration within the diffusion volume does not change with respect to time. And from the definition “flux” itself, we also have
\[
\mathcal{F}_x = \rho \frac{\partial x}{\partial t} ,
\]
\[
\mathcal{F}_y = \rho \frac{\partial y}{\partial t} .
\]  
If the flux of \(\rho\) is divided by the value of \(\rho\) at a location \((x, y)\), the diffusive transport velocity of \(\rho\) is
\[
\mathbf{v} = \frac{\mathcal{F}}{\rho} = -D \nabla \frac{\rho}{\rho} .
\]  
(5.2)

In diffusion a cell migrates from an initial location to its final equilibrium location via a very non-direct route. This route can be captured by a velocity function that gives the velocity of a cell at every location in the route for a given time \(t\). This velocity at certain position and time is determined by the local density gradient and the density itself. Intuitively, sharp density gradient or low density will cause cells to move faster. We can define a velocity field \(\mathbf{v}(x, y) = (v_x(x, y), v_y(x, y))\) of diffusion at time \(t\), which can be computed as:
\[
v_x(x, y, t) = -\frac{D \frac{\partial \rho}{\partial x}}{\rho(x, y)} ,
\]  
(5.3)
\[
v_y(x, y, t) = -\frac{D \frac{\partial \rho}{\partial y}}{\rho(x, y)} .
\]  
(5.4)
Because cells can not move out of boundary, the boundary condition is
\[ v(x, y) \cdot \vec{n} = 0, \]
\[ \Rightarrow \nabla \rho(x, y, t) \cdot \vec{n} = 0, \quad \text{for } (x, y) \text{ at } \partial \Omega, t > 0, \]
Where, \( \vec{n} \): the unit vector with normal direction to the boundary.

Therefore, starting from an initial location \((x(0), y(0))\), the cell location \((x(t), y(t))\) at time \(t\) can be calculated by integrating the velocity field,
\[ x(t) = x_0 + \int_0^t v_x(x(t'), y(t'), t')dt', \quad (5.6) \]
\[ y(t) = y_0 + \int_0^t v_y(x(t'), y(t'), t')dt', \quad (5.7) \]

With (5.3-5.4), (5.6-5.7), we can incrementally change a placement based on the continuous density distribution. However, the continuous form of these equations can not be directly used in placement migration, which has a discrete grids structure. Then, in next section, we will give a discrete approximation for these equations.

### 5.3 Numerical simulation of diffusion process

#### 5.3.1 Discretization

In chapter 4, we partition the placement domain into grid cells, and use grid coordinates \((i\Delta, j\Delta)\) instead of the continuous coordinates \((x, y)\). Here, similarly, we set the placement domain \(\Omega = [0, 1]^2\). We want to compute a numerical solution of velocity field, consequently, we choose a time step \(\Delta t > 0\) and a grid size \(\Delta x = \Delta y = \Delta = 1/N\), and define the following grids in \([0, 1] \times [0, \infty)\):
\[ x_i = (i - 1)\Delta x = (i - 1)\Delta, \quad (i = 1, 2, \ldots, N + 1), \]
\[ y_j = (j - 1)\Delta y = (j - 1)\Delta, \quad (j = 1, 2, \ldots, N + 1), \]
\[ t^n = n\Delta t, \quad (n = 0, 1, 2, \ldots). \]

The approximation of \(\rho(x_i, y_j, t^n)\) is denoted by \(\rho_{i,j}^n\). The way in calculating the densities on grid points is same as that in the previous algorithm (see chapter 4: 4.3.1).

Then, Assume that the density \(\rho_{i,j}^n\) has already been computed for time \(n\). Next, we need to find how the density changes and cells move during the next time step \(n + 1\). After we compute the \(\rho_{i,j}^n\), then we can compute the velocity
of the point which is located in the center between 2 grid points by discretizing (5.3)-(5.4),

\[
v^n_{x(i-1/2,j)} = -\frac{D}{\Delta} \frac{\rho^n_{i,j} - \rho^n_{i-1,j}}{\rho^n_{i-1/2,j}} ,
\]

\[
v^n_{y(i,j-1/2)} = -\frac{D}{\Delta} \frac{\rho^n_{i,j} - \rho^n_{i,j-1}}{\rho^n_{i,j-1/2}} .
\] (5.8)

And from the boundary condition (5.5),

\[
v_x(x, y, t^n) = 0, \quad \text{for } (x, y) \text{ at vertical boundary },
\]

\[
v_y(x, y, t^n) = 0, \quad \text{for } (x, y) \text{ at horizontal boundary} .
\] (5.9)

To get the new placement, we need to compute the integral equation (5.6)-(5.7). We use the discretized form \((x(t^n), y(t^n))\) instead of \((x(t), y(t))\) to represent the location of a cell. Then (5.6)-(5.7) can be transformed into a recursive form, that is, in each iteration, after the placement migrates, we assume all cells stop at the new coordinates. Suppose we have already computed \((x(t^n), y(t^n))\), then we can easily get the coordinate of a cell after one time step.

\[
x(t^{n+1}) = x(t^n) + v_x(x(t^n), y(t^n), t^n) \Delta t ,
\]

\[
y(t^{n+1}) = y(t^n) + v_y(x(t^n), y(t^n), t^n) \Delta t .
\] (5.10)

Generally, we formulate the new placement at the moment \(n + 1\) in the following form:

\[
X^{n+1} = X^n + \mathbf{v} \Delta t ,
\] (5.11)

where, \(X\) is defined in section 2.2.

\(X_i \in X, X_i = (x_i, y_i)^T \) for \(i = 1, 2, \ldots, n\).

\[
\mathbf{v} = \begin{pmatrix}
v_{x1} & v_{x2} & \cdots & v_{xn} \\
v_{y1} & v_{y2} & \cdots & v_{yn}
\end{pmatrix} .
\]

5.3.2 Velocity Interpolation

In section 5.3.1, we calculate the velocity on the grid points with eq.(5.8). One problem here is every cell within a grid has different velocity and will get the different displacement. We apply bi-linear interpolation to assign the velocity to every cell. This approach is the same as calculating the gradient of potential in chapter 4 (see section 4.3.3). Figure 5.1 illustrates the velocity interpolation situation for one cell.

From (5.8), we can calculate the x-direction velocity at mid-point \((i - 1/2, j)\) between grid points \((i, j)\) and \((i - 1, j)\). Similarly, y-direction velocity can be obtained in the same way. Then we determine in which box with four velocities one cell is located. The interpolation process will be applied twice for computing \(v_x\) and \(v_y\) for one cell.
5.3.3 The magnitude factor

For equation (5.10), we can rewrite it in the following form

$$X^{n+1} = X^n + \Delta X^n,$$

$$\Rightarrow \Delta X^n = v^n \Delta t .$$ (5.12)

We introduce a magnitude factor $\alpha$ into the displacement in order to control the speed of cells spreading,

$$\Delta X = \alpha \cdot v = \alpha \left( - \frac{\nabla \rho}{\rho} \right) .$$ (5.13)

Here, the velocity field $v = -\nabla \rho$, the diffusivity $D$ is not considered. The speed of diffusion will be taken care by the magnitude factor $\alpha$.

To determine the size of the magnitude factor we will choose, first the dimension of $\alpha$ should be taken account into. from equation (5.13), balancing the dimension of both sizes of this equation, we have

$$l \propto \alpha \cdot \frac{1}{l} ,$$

$$\Rightarrow \alpha \propto l^2 , \quad l : \text{ the length unit} .$$ (5.14)

Relating (5.13) and (5.14), we find

$$\Delta X = v \Delta t = -D \Delta t \frac{\nabla \rho}{\rho} = -\alpha \frac{\nabla \rho}{\rho} ,$$

$$\Rightarrow \alpha = D \Delta t .$$

When we examine 5.13 again, we find that $\alpha$ could be equal to $C \Delta^2$, $\Delta = 1/N$ is the grid size when we discritize the placement domain. $C$ is a constant.
5.3.4 The smoothed density

We calculate the velocity field by using (5.8). Notice that the initial density distribution might be not so smooth, or we can say some regions are crowded, this will results in sharp density difference around the border of these crowded regions. In order to apply the numerical smoothing quality of diffusion and relaxation found in local process, we expect the gradient of density field more even to obtain a smooth cells movement. Therefore, this smoothed density field will give us more freedom in choosing the magnitude factor.

To make the density distribution more smooth, we introduce a “smooth function”. This function use some weights to relate the local density with the neighbor region densities:

\[
\hat{\rho}_{i,j} = \gamma_1 \rho_{i-1,j+1} + \gamma_2 \rho_{i,j+1} + \gamma_3 \rho_{i+1,j+1} + \gamma_4 \rho_{i-1,j} + \beta \rho_{i,j} + \gamma_5 \rho_{i+1,j} + \gamma_6 \rho_{i-1,j-1} + \gamma_7 \rho_{i,j-1} + \gamma_8 \rho_{i+1,j-1},
\]

where, \(\hat{\rho}_{i,j}\) is the smoothed density for \(\rho_{i,j}\), \(\gamma_1, \gamma_2, \ldots, \gamma_8\) are the weights for the densities of 8 grid points which are around grid point \((i, j)\). \(\beta\) is the weight for \(\rho_{i,j}\).

5.4 Experiments

5.4.1 Small cases in the diffusion process

With this diffusion based algorithm for cell spreading, we obtain the new placement by calculating the velocity of every cell. Such velocity field is based on the present density distribution, and equals the flux of \(\rho\) divided by the value of \(\rho\) at a local location \((x, y)\). Under this diffusion process principle, some questions arise for us: what is the expected final result placement? What's the migration trajectory of every cell?

The final equilibrium state will be an equal concentration(density) distribution. For each cell, there will be no density difference around it. From (5.8), if the gradient of density is zero, then we obtain a zero velocity field. Using the velocity interpolation rule, we assign the velocity to every cell, which results in an unchange new placement.

To show this diffusion process resulted in an equilibrium state, we list 2 small placement cases. In the final state, the arrow represents the velocity vector which is pointing from the initial coordinate to the final coordinate.
**1 cell:** after 3 iterations

![Figure 5.2: The migration trajectory of 1 cell placement in chapter 5](attachment:image)

**25 cells:** after 190 iterations.

![Figure 5.3: The migration trajectory of 20 cell placement in chapter 5](attachment:image)

### 5.4.2 Application examples

We choose a testing case with 1600 cells to implement this diffusion based algorithm. In this chapter, we still use the maximum scaled density as the program stop criterion. Table 6.1 lists the given data of our testing model. The explanations of these notations are

- \( n \): the number of cells
- \( R(\Omega) \): the average density over domain
- \( N_w \): the number of windows
- \( N \): the number of grid cells on each dimension
- \( \text{iter}_{\text{max}} \): the maximum iteration number
- \( \rho_{sM} \): the maximum scaled density parameter
width = height

width = height

\( n \)  \( R(\Omega) \)  \( N_w \)  \( N \)  \( \text{iter}_{\text{max}} \)  \( \text{stop criterion: } \rho_{s\text{M}} \)

\begin{array}{cccccccc}
0.025 & 1600 & 1.0 & 16 & 80 & 10000 & 1.06
\end{array}

Table 5.1: Parameters of the testing model in chapter 5

Figure 5.4 shows the initial placement and the initial density of window distribution. We set 16 windows for this example, and the size of the window is 100 times of the size of every cell.

\[ x \]
\[ y \]

\[ x \]
\[ y \]

The initial placement

Figure 5.4: The initial placement (in chapter 5)

We choose the number of grid cells in each dimension \( N = \frac{1}{4 \text{width}} = 2/0.025 = 80 \). And we set the stop criterion of the algorithm as: the maximum window scaled density \( \rho_{s\text{max}} = 1.06 \) or iteration number equals \( \text{iter}_{\text{max}} \). From section 5.3.3, we know the magnitude factor should be chosen as \( D \Delta t \) that equals \( C \Delta^2 \). For simplification, we set the diffusion coefficient \( D = 1 \), then the magnitude factor \( \alpha = \Delta t = C \Delta^2 \). We apply the smooth function 5.15 (see section 5.3.4) for this application example, the results under vary \( \alpha \) is listed in table 5.2. For comparison, the results with non-smoothed test example also are listed in table 5.2.

\[
\begin{array}{cccccc}
\Delta t & \alpha & \text{iter} & \text{distance} & s & \text{iter} & \text{distance} & s \\
\frac{1}{N^2} & \frac{1}{N^2} & 1791 & 3.929816 & 0.015293 & 2000 & 3.781333 & 0.028872 \\
\frac{1}{N^2} & \frac{1}{N^2} & 900 & 3.941216 & 0.015005 & 1592 & 3.974455 & 0.021813 \\
\frac{1}{N^2} & \frac{1}{N^2} & 453 & 3.945664 & 0.014644 & 652 & 3.947433 & 0.023053 \\
\frac{1}{N^2} & \frac{1}{N^2} & 224 & 3.930875 & 0.015620 & 305 & 4.538958 & 0.014858 \\
\frac{1}{N^2} & \frac{1}{N^2} & 133 & 4.778874 & 0.015308 & 123 & 8.818103 & 0.019875 \\
\end{array}
\]

Table 5.2: The results of experiment example \((\alpha = D \Delta t, D = 1)\)

From table 5.2, the iteration times is inverse linear with the magnitude fac-
tor α. When we increase α two times as the previous one, the iteration times is almost half of the iteration times in the previous case. Because we use the magnitude factor to control the displacement(5.13), the velocity in every iteration is calculated by the gradient of density field divided by the local density, then the velocities for different α in each iteration are almost the same, thus increasing the magnitude factor will result in the iteration times decreasing linearly. From this table, we can see that for this testing problem $\alpha = \frac{2}{N^2}$ is the best choice considering the algorithm runtime. Besides, from the result figures(see figure 5.5-5.7), there’s not much difference in these results except for the iteration times.

![Figure 5.5: The maximum scaled density of window in each iteration. $\alpha' = \frac{2}{(N^2)$: in non-smoothed test example](image)

**The experiment with $\alpha = \frac{4}{N^2}$**

The program reaches to the stop criterion $\rho_{\text{st}}$ very fast but unstable, as we see the results in figure 5.5 and figure 5.6. The total movement distance is much larger than that of other cases. And from the result placement figure, we see the cells are not distributed fully over the placement domain. The data from the density of grid points distribution is higher than that of other cases(see figure 5.9), and is far from the ideal density of grid points ($\rho = 1.0$).

**The non-smoothed experiment cases**

We also list the results of experiments without smoothed density function to compare with the previous results with the smoothed density function. For $\alpha = \frac{2}{N^2}$, the cells are not spread fully over the placement domain as we see in the result placement figure(see figure 5.2). Also algorithm in this experiment case is unstable, as shown in figure 5.5-5.7. The total movement distance of each case (with different $\alpha$) is larger than the corresponding smoothed case.
5.5 Conclusions

The diffusion based approach generates cell velocities from local density distribution and use integral equations of velocities to directly compute cell location. The main advantages to this approach are

- It spreads the placement smoothly which is more likely to preserve the integrity of the original placement.

- It changes the placement incrementally and the runtime has a inverse linear relation with the magnitude factor $\alpha$, thus we can change $\alpha$ to gain the desire spread of cell spreading.

- Its implementation in programming is less complicated compared with the force-based algorithm in chapter 4.

However, also we compare this algorithm with the force-based algorithm, implementing it to attain the expected results will take more iterations in running the program.
Figure 5.6: The even distribution measurement with different magnitude in each iteration. $\alpha' = 2/(N^2)$: in non-smoothed test example

Figure 5.7: The movement distance with different magnitude factor. $\alpha' = 2/(N^2)$: in non-smoothed test example
Figure 5.8: The result placement with different $\alpha$ (in chapter 5)
Figure 5.9: The density distribution with different $\alpha$ (in chapter 5)
Chapter 6

Comparison experiments

We already discussed three cell spreading algorithms in the previous chapters. In this chapter, some results we obtained on running these algorithms for different sets of values. We studied the algorithms by comparing their performance based on the final results of the placement and the number of iterations taken. Graphical visualizations of these results were also generated which helped in analyzing the quality of the placement and the shortcomings of the algorithms.

6.1 The testgroups

All three algorithms were run using the same initial placements which is generated by using the placement model in section 2.5. We use 5 initial placement models for comparison. The size of placement is from 100 cells to 2500 cells. Figure 6.1 shows the initial placement with 1600 cells.

Figure 6.1: The initial placement of the test case with \( n = 1600 \)
Table 6.1 lists the given data for the initial placement and the stop criterion for running the program. Here, we choose the extreme occupation placement model in which $R(\Omega) = 1.0$. The purpose of choosing such initial placement with high density distribution is to verify whether the algorithm is good enough to handle high density placement problem ($R(\Omega) > 0.95$). And we choose a certain number of windows for each test group such that the size of window is 25 times of the size of the cell. The notations in table 6.1 are:

- $n$: the number of cells
- $R(\Omega)$: the average density over domain
- $N_w$: the number of windows
- $\text{iter}_{\text{max}}$: the maximum iteration number
- $\rho_{sM}$: the maximum scaled density

<table>
<thead>
<tr>
<th>test</th>
<th>width = height</th>
<th>$n$</th>
<th>$R(\Omega)$</th>
<th>$N_w$</th>
<th>$\text{iter}_{\text{max}}$</th>
<th>stop criterion: $\rho_{sM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>100</td>
<td>1.0</td>
<td>4</td>
<td>2000</td>
<td>1.01</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
<td>400</td>
<td>1.0</td>
<td>16</td>
<td>2000</td>
<td>1.01</td>
</tr>
<tr>
<td>3</td>
<td>$1/30$</td>
<td>900</td>
<td>1.0</td>
<td>36</td>
<td>2000</td>
<td>1.01</td>
</tr>
<tr>
<td>4</td>
<td>0.025</td>
<td>1600</td>
<td>1.0</td>
<td>64</td>
<td>2000</td>
<td>1.01</td>
</tr>
<tr>
<td>5</td>
<td>0.02</td>
<td>2500</td>
<td>1.0</td>
<td>100</td>
<td>2000</td>
<td>1.01</td>
</tr>
</tbody>
</table>

Table 6.1: The test groups of the initial placement

The following list is the abbreviation of three algorithms:

- algorithm 1: pair-movement algorithm
- algorithm 2: force-based algorithm with electrostatic model
- algorithm 3: diffusion-based algorithm

<table>
<thead>
<tr>
<th>algorithm</th>
<th>$\alpha$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6</td>
<td>$\frac{1}{0.6 \times \text{width}}$</td>
</tr>
<tr>
<td>2</td>
<td>0.6</td>
<td>$\frac{1}{0.6 \times \text{width}}$</td>
</tr>
<tr>
<td>3</td>
<td>$2/N^2$</td>
<td>$\frac{1}{0.6 \times \text{width}}$</td>
</tr>
</tbody>
</table>

Table 6.2: The input parameters of algorithms ($\alpha$ is the magnitude factor which is used to control the speed of cells’ movement. $N$ is the number of grid cells on each dimension for algorithm 2 and algorithm 3 when we discretize the placement domain)

For each algorithm, we also need to input some parameters (see table 6.2). These parameters of each algorithm are chosen based on the experiments results of the previous chapters.
6.2 Experimental Results

All three algorithms are implemented in Matlab. We compare the performance of the algorithms mainly in three aspects: the runtime of the algorithm (iteration times), the total movement distance and the even distribution measurement. Table 6.3 lists the data of the final result placement. The results of algorithm 2 and algorithm 3 are also illustrated in figure 6.2.

We see that from the data in table 6.3, the results (distance, s, \( \max(\rho_{w(i,j)}) \)) with algorithm 2 are the smallest, and the program takes the least iterations to attain the stop criterion \( \rho_s M \). Algorithm 2 generates in better placement in all cases, but the results are not much different from those of algorithm 3.

Another view of the difference among these 3 algorithms is from the result figures (see table 6.3). We list the result figures of test 4 with \( n = 1600 \). In the first line, three graphs show the maximum scaled density of window in every iteration corresponding to 3 algorithms respectively. For algorithm 1, the maximum scaled density of window decreases fast during first 100 iterations, then oscillates sharply until it reaches to a lowest point at iteration 500. After that it increases to a certain high number. This is because the cell's movement in this algorithm is just in 2 (vertical and horizontal) directions. When we removed the overlaps at the best we can, some cells are still stacked together in some crowded regions, then these cells will oscillated and can never be moved apart. So at this moment (iteration 500), \( \max(\rho_{w(i,j)}) \) increases and tends to

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>n</th>
<th>iteration</th>
<th>Distance</th>
<th>s</th>
<th>( \max(\rho_{w(i,j)}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>2000</td>
<td>1.797475</td>
<td>0.033953</td>
<td>1.096894</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>2000</td>
<td>3.460737</td>
<td>0.036685</td>
<td>1.118937</td>
</tr>
<tr>
<td></td>
<td>900</td>
<td>2000</td>
<td>4.882584</td>
<td>0.057833</td>
<td>1.345474</td>
</tr>
<tr>
<td></td>
<td>1600</td>
<td>2000</td>
<td>5.349228</td>
<td>0.065744</td>
<td>1.466422</td>
</tr>
<tr>
<td></td>
<td>2500</td>
<td>2000</td>
<td>6.143436</td>
<td>0.078676</td>
<td>2.075206</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>5</td>
<td>1.664774</td>
<td>0.002221</td>
<td>1.008885</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>21</td>
<td>3.286138</td>
<td>0.002562</td>
<td>1.009713</td>
</tr>
<tr>
<td></td>
<td>900</td>
<td>140</td>
<td>5.078362</td>
<td>0.002381</td>
<td>1.009867</td>
</tr>
<tr>
<td></td>
<td>1600</td>
<td>447</td>
<td>5.415719</td>
<td>0.001492</td>
<td>1.009975</td>
</tr>
<tr>
<td></td>
<td>2500</td>
<td>302</td>
<td>6.740909</td>
<td>0.001855</td>
<td>1.009978</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>44</td>
<td>1.693493</td>
<td>0.004389</td>
<td>1.008951</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>394</td>
<td>3.315710</td>
<td>0.002077</td>
<td>1.009955</td>
</tr>
<tr>
<td></td>
<td>900</td>
<td>812</td>
<td>5.092172</td>
<td>0.002106</td>
<td>1.009935</td>
</tr>
<tr>
<td></td>
<td>1600</td>
<td>1601</td>
<td>5.429430</td>
<td>0.001862</td>
<td>1.009951</td>
</tr>
<tr>
<td></td>
<td>2500</td>
<td>1810</td>
<td>6.743475</td>
<td>0.001945</td>
<td>1.009990</td>
</tr>
</tbody>
</table>

Table 6.3: The results of the comparison experiments
be a fix number. These cells’ oscillations are so small that the result placement doesn’t change much, then the even distribution $s$ and the total movement distance $d$ will stagnate. Algorithm 2 converges very fast, as we see in those three result figures. For algorithm 3, in the figure of $\max(\rho_{\text{sw}(i,j)})$ and the figure of $s$, the oscillation appear in the final stage (after iteration 600). The reason is we choose a certain high value for the magnitude factor $\alpha = 2/N^2$. If we set it a lower value, the curves will decrease more smoothly. The total run-time in Matlab is different for these 3 algorithms. For example, when we do the test with $n = 2500$, with algorithm 2, the run-time is around 2 minutes while with algorithm 3, the run-time is about 6 minutes. For algorithm 1, it never attains the stopping criterion: $\rho_{sM}$, and it is more slower than the other 2 algorithms.

We also notice that although algorithm 2 yields better solution, it requires more effort to fulfill the algorithm of itself. In matlab, the compiler uses the optimal package to solve the singular linear system for algorithm 2, whereas algorithm 3 yields also “good” result placement in relatively small effort. Therefore, to really compare algorithm 2 and algorithm 3, we need to analyse their computational complexity.

### 6.3 Complexity analysis

The computational cost, or complexity, of numerical algorithms is usually measured by the number of arithmetic operations required. Traditionally, numerical analysts have counted by multiplications (and possibly divisions and square roots) [10].

In algorithm 2, we first solve the related boundary problem (4.11) by using the numerical method (Finite difference method), that is reduce this BVP into a linear system. And we solve this linear system in a direct way with LU-decomposition (see section 4.3.2). Implementing this algorithm, it demands compute the solution $\Phi$ of the linear system in every iteration as the density distribution is changed from the result placement of the previous iteration. In other words, we should compute $x = S(U + R)^{-1}L^{-1}PSb$ (see equation (4.26)) in every iteration. Then the complexity of this computation in every iteration is $O(N^{3/2})$, here, the size of the corresponding linear system is $N \times N$.

For algorithm 3, the velocity field is computed by equation (5.2) with the boundary condition (5.5), therefore we can calculate the velocity field directly with the density field in every iteration and don’t need to solve any linear system. The velocity of every cell is obtained by using the bi-linear interpolation with the velocity field (section 5.3.2). Then we say the complexity of computing the velocity field in every iteration is linear with the number of the cells. This also can be seen from figure 6.2(a).
If the placement domain $[0, 1]^2$ is fully occupied by cells, assume that the total number of standard cells is $n^2$, and $n = \frac{1}{\text{width}}$. For the discretization of the BVP (4.11) in algorithm 2, $(N-1) = \frac{1}{0.5 \times \text{width}} = 2n$, then the complexity of algorithm 2 in every iteration equals $O(N^{3/2}) = O((2n + 1)^{3/2})$. If the number of cells is large, for every iteration, the computational cost of algorithm 2 will be very expensive. From the above analysis, it is easy to conclude that the complexity in every iteration of algorithm 3 is less than that of algorithm 2.

6.4 Conclusions

We present the results of the comparison tests on three algorithms for cell spreading in this chapter. Here, we summarize the characters of each algorithm into viewpoints (advantage and disadvantage):

<table>
<thead>
<tr>
<th>algorithm</th>
<th>advantage</th>
<th>disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>alg 1</td>
<td>the motion is predictable, emphasis on reducing the overlaps</td>
<td>not efficient in spreading</td>
</tr>
<tr>
<td>alg 2</td>
<td>spreading speed is the most fast, attains the stop criterion in few iterations</td>
<td>complicated in computation</td>
</tr>
<tr>
<td>alg 3</td>
<td>spread cells smoothly, easy to be implemented</td>
<td>converges slowly</td>
</tr>
</tbody>
</table>

Table 6.4: Comparison summary of three cell spreading algorithms

The cell spreading speed can be controlled by changing the magnitude factor $\alpha$. For algorithm 1 and algorithm 2, $\alpha$ is dimensionless. For algorithm 3, $\alpha$ with dimension in length unit $l^2$ And more complicated to determine the "good" choice.

If average density $R(\Omega)$ is low, namely $0.6 \leq R(\Omega) \leq 0.8$, using algorithm 2 or algorithm 3, the cells can be really moved apart and no overlaps left. We didn't list the related results for such case in this chapter. From another angle of this point, there's a trick to think about the placement with low $R(\Omega)$: first, we can use the cell spreading algorithms to spread cells with a fully occupied placement example ($R(\Omega) = 1.0$), after the expected result placement is obtained, then we can reduce the cell's size such that the average density $R(\Omega)$ is satisfied with the practical need.
Figure 6.2: Comparison results with algorithm 2 and algorithm 3
Figure 6.3: Comparison of result figures (n = 1600)
Chapter 7

Summary

This thesis investigates three cell spreading algorithms for the VLSI detailed placement. Traditionally, the VLSI placement includes two steps: the global placement and the detailed placement. We introduced a new step cell spreading before the detailed placement. The cell spreading problem arises from the motivation to spread cells further after the global placement while keeping the generic placement properties (cells ordering, wirelength) as much as possible. The problem is transferred into looking for a cell spreading method which results to be very easy to incorporate in a complete design flow. Moreover, this approach should be very efficient if implemented in iterative schemes and it can give better insight for designers to select the best combination of parameters for cell spreading problem.

This thesis focuses on two main goals for cell spreading:

- Even distribution
- Minimal movement

The experimental results are presented for comparison with different control parameters. And those results have been examined according to these two objectives for different algorithms in this thesis.

Chapter 2 has illustrated the cell spreading situation in mathematical way; given possible notations and definitions for cell spreading process. We set up a testing placement model in normal random distribution. And the cell spreading tool has been described in the last part of this chapter.

Three cell spreading algorithms have been introduced in the following three chapters respectively. The pair-movement algorithm (see chapter 3) comes from the intuition idea of moving apart the overlapping cells. The movement cost depends on the overlapping parts between every pair cells, that’s also the meaning for the name of this algorithm itself. Because the cell’s movement is just in two directions: horizontal and vertical, the operation to change the initial
placement is just to spread out cells locally, then this algorithm is not efficient in even distribution.

Chapter 4 presented a force-based algorithm with electro-static model. Relating electro-static force and the density field, we derived a boundary value problem (4.11) which comprise a poisson equations and a symmetric boundary condition. Using the finite central difference method, the discretized poisson problem has been written as a linear singular system. The force is just the gradient of the potential which is the solution of this linear singular system. The force pushed on every cell will result in the placement migration.

In chapter 5, the movement of cells is simulated as the flow due to gradient of density. This diffusion based algorithm generates cell velocities from local density distribution and use the integral equations of velocities to directly compute cell location. Based on this point, this algorithm spreads cells smoothly and is easy to implement in programming.

The results of the comparison experiments on these three algorithms are discussed in chapter 6. From these results, we found algorithm 1 (pair-movement algorithm) is not good for cell spreading because the purpose for cell movement is not for even distribution. The experimental results of algorithm 2 and algorithm 3 are comparable. The algorithm 2 converges very fast while the computational cost of it in every iteration is higher than that of algorithm 3.

Possible future work could be further investigation of the magnitude factor. In this thesis, various magnitude factors are chosen in experiments for each algorithm. We hope the algorithm more intelligent that the magnitude factor can be chosen automatically inside the algorithm. For every cell, the magnitude factor could be different and just depends on the local density distribution.

Another point is from the real design problem. We could extend the square testing placement domain to non-square placement domain. Also, we can change the standard cells with different width.

For readers who are interested in Algorithm 1 ("Pair-movement algorithm") could try changing it into a symmetric algorithm in which both cells are moved half the overlapping distance instead of moving one cell with the whole overlapping distance.
Bibliography


