Synthesis of Digital Microfluidic Biochips with Reconfigurable Operation Execution

Maftei, Elena

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Synthesis of Digital Microfluidic Biochips with Reconfigurable Operation Execution

Elena Maftei

Kongens Lyngby 2011
IMM-PHD-2011-257

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

Microfluidic biochips are an alternative to conventional biochemical laboratories, and are able to integrate on-chip all the necessary functions for biochemical analysis. The "digital" biochips are manipulating liquids not as a continuous flow, but as discrete droplets on a two-dimensional array of electrodes.

The main objective of this thesis is to develop top-down synthesis techniques for digital microfluidic biochips. So far, researchers have assumed that operations are executing on virtual modules of rectangular shape, formed by grouping adjacent electrodes, and which have a fixed placement on the microfluidic array.

However, operations can actually execute by routing the droplets on any sequence of electrodes on the biochip. Thus, we have proposed a routing-based model of operation execution, and we have developed several associated synthesis approaches, which progressively relax the assumption that operations execute inside fixed rectangular modules.

The proposed synthesis approaches consider that i) modules can dynamically move during their execution and ii) can have non-rectangular shapes. iii) We have relaxed the assumption that all electrodes are occupied during the operation execution, by taking into account the position of droplets inside modules. Finally, iv) we have eliminated the concept of virtual modules and have allowed the droplets to move on the chip on any route. In this context, we have also shown how contamination can be avoided.

We have extensively evaluated the proposed approaches using several real-life case studies and synthetic benchmarks. The experiments show that by considering the dynamically reconfigurable nature of microfluidic operations, significant improvements can be obtained, decreasing the biochemical application completion times, reducing thus the biochip area and implementation costs.

Resumé

Lab-on-a-chip–et komplet biokemisk laboratorium på en mikrochip, også kaldet en biochip - er et lovende alternativ til konventionelle biokemiske laboratorier. "Digitale" biochips arbejder med væsker, ikke som en kontinuerlig strøm, men som diskrete dråber på et todimensionelt gitter af elektroder.

Hovedmålet med denne afhandling er at udvikle top-down syntesetekniker til digitale mikrofluidiske biochips. Indtil videre har forskere antaget, at operationer bliver udført på virtuelle, rektangulære moduler, der bliver dannet ved at gruppere naboelektroder, og som har en fast placering på elektrodegitteret.

Dog kan operationer faktisk blive udført ved at dirigere dråberne på en hvilken som helst sekvens af elektroder på biochipen. Derfor har vi foreslået en rutebaseret model for udførelse af operationer, og vi har udviklet adskillige tilhørende syntesemetoder, som progressivt går væk fra antagelsen om, at operationer bliver udført af fastsatte, rektangulære moduler.

De foreslåede syntesemetoder tager i betragtning, at i) moduler kan flyttes rundt dynamisk mens operationerne bliver udført og ii) moduler kan have ikke-rektangulære former. iii) Vi er gået væk fra antagelsen om, at alle elektroder i modulet er i brug under udførelse af operationen ved at tage placeringen af dråber internt i modulerne i betragtning. Endeligt iv) er vi gået væk fra konceptet med virtuelle moduler and har tilladt dråberne at bevæge sig frit rundt på chipen. I denne sammenhæng har vi også vist, hvordan kontaminering kan undgås.

Vi har evalueret de foreslåede metoder ekstensivt ved brug af adskillige cases og syntetiske benchmarks. Eksperimenter viser, at betydelige forbedringer kan opnås ved at tage i betragtning, at biochips og tilhørende applikationer er af en natur, der tillader og endda lægger op til dynamisk omkonfigurerbarhed. Herved kan biokemiske applikationer udføres hurtigere, hvilket reducerer størrelsen på biochipsne og derved prisen på en chip.

Preface

This thesis was prepared at Informatics and Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering.

The thesis deals with optimization tools for the synthesis of digital microfluidic biochips, in an effort to offer the microfluidic community with the same level of support present in the semiconductors industry.

The work has been supervised by Associate Professor Paul Pop and co-supervised by Professor Jan Madsen.

Lyngby, April 2011

Elena Maftei

Papers included in the thesis

- Elena Maftei, Paul Pop, and F. Popenţiu Vlădicescu. Synthesis of Reliable Digital Microfluidic Biochips using Monte Carlo Simulation. *Proceedings of the European Safety and Reliability Conference*, pp. 2333–2341, 2008. Published.
- Elena Maftei, Paul Pop, Jan Madsen, and Thomas Stidsen. Placement-Aware Architectural Synthesis of Digital Microfluidic Biochips using ILP. *Proceedings of the International Conference on Very Large Scale Integration of Systems on Chip*, pp. 425–430, 2008. Published.
- Elena Maftei, Paul Pop, and Jan Madsen. Tabu Search-Based Synthesis of Dynamically Reconfigurable Digital Microfluidic Biochips. *Proceedings of the Compilers, Architecture, and Synthesis for Embedded Systems Conference*, pp. 195–203, 2009. Published. Best Paper Award.
- Elena Maftei, Paul Pop, and Jan Madsen. Tabu Search-Based Synthesis of Digital Microfluidic Biochips with Dynamically Reconfigurable Non-Rectangular Devices. *Journal of Design Automation for Embedded Systems*, 14(3), pp. 287–308, 2010. Published. Selected as part of the Special Issue with the Best Papers from Embedded Systems Week 2009.
- Elena Maftei, Paul Pop, and Jan Madsen. Routing-Based Synthesis of Digital Microfluidic Biochips. *Proceedings of the Compilers, Architecture, and Synthesis for Embedded Systems Conference*, pp. 41–49, 2010. Published. Best Paper Award Candidate.
- Elena Maftei, Paul Pop, and Jan Madsen. Module-Based Synthesis of Digital Microfluidic Biochips with Droplet-Aware Operation Execution. Under review in *Journal on Emerging Technologies in Computing Systems.*

Acknowledgements

To all the people that inspired me throughout my life. Thank you!

Contents

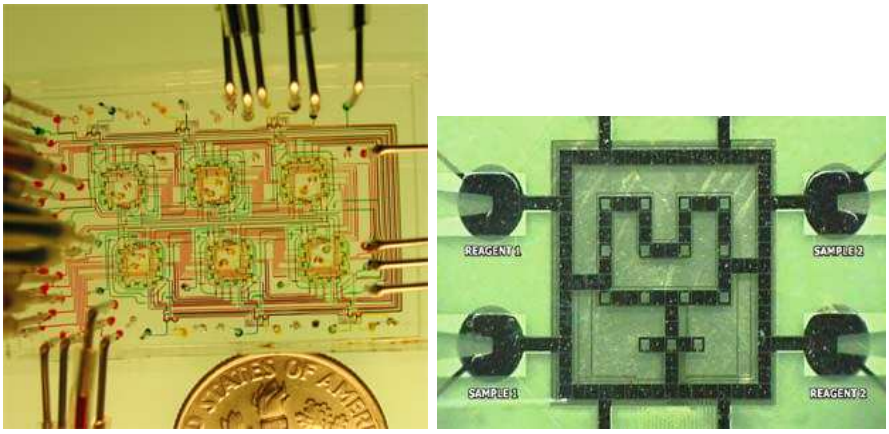
Chapter 1

Introduction

According to "Moore's law" [39] the number of transistors on an integrated circuit doubles approximately every two years. "More than Moore" explores new applications in which such systems can be used, focusing on function diversification rather than increasing density. An emerging field related to embedded systems is the design of efficient, low-cost devices for the bio-medical area, which has been highlighted by the International Technology Roadmap for Semiconductors 2007 [24] as an important system driver for the near-future [4].

The history of such devices, called biochips (also referred to as lab-on-chips), started in the late 1980's, being strongly connected to the progresses done in genomics. The possibility of analyzing and amplifying deoxyribonucleic acid (DNA) fragments lead to the development of DNA microarrays, two-dimensional arrays of biosensors on which genetic tests can be performed. On such devices thousands of biosensors (DNA fragments) are affixed to a substrate (typically glass or silicon) using photolithography or ink-jet printing, and their hybridization process with fragments of target DNA is analyzed. DNA microarrays have many applications, including genotyping, mutation analysis, disease diagnosis and drug discovery [52].

The advances in genomics and the growing interest in biological systems have lead in the late 1990s to the manufacturing of protein arrays. These devices are created using a similar technology to DNA arrays, with thousands of proteins



(a) Continuous-flow microfluidic biochip [44] (b) Digital microfluidic biochip [60]

Figure 1.1: Microfluidic biochips

being immobilised on a substrate and being exposed to different molecules (e.g., other proteins, peptides). As a result, the amount of specific proteins in biological samples (e.g., blood) can be measured, which is particularly important in fields such as clinical diagnosis and drug discovery [21].

A further step in the development of miniaturized laboratories has been the creation of microfluidic biochips, on which biochemical reactions involving liquids can be performed. Such devices are able to integrate on-chip all the necessary functions for biochemical analysis such as, transport, splitting, merging, dispensing, mixing, and detection [11], using very small amount of fluids (micro- or nanoliters).

There are two types of microfludic biochips. The first type is based on the manipulation of continuous liquid through fabricated micro-channels, using external pressure sources or integrated mechanical micro-pumps [63], see Figure 1.1a. Although initially used for simple biochemical applications due to their complexity, the advances made in soft lithography fabrication techniques have lead to the microfluidic large-scale integration. This technology aims at increasing significantly the number of assays that can be performed concurrently by integrating on the chip hundreds to thousands of micro-mechanical valves and control components [35].

The second type is based on the manipulation of discrete, individually controllable droplets on a two-dimensional array of identical cells, see Figure 1.1b. The actuation of droplets is performed without the need of micro-structures, by using software-driven electronic control [4]. This type is also referred to as “digital

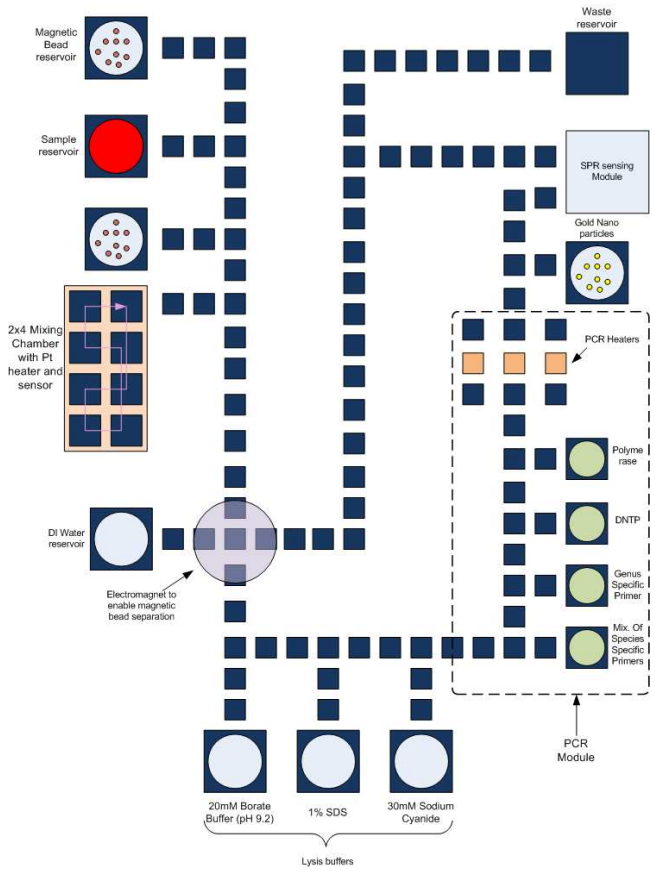


Figure 1.2: Digital biochip for malaria detection [8]

microfluidics", due to the analogy between the droplets and the bits in a digital system.

Biochips offer a number of advantages over conventional biochemical procedures. The main problem in performing biochemical applications is the high cost of reagents. By handling small amount of fluids, biochips provide higher sensitivity while decreasing the reagent consumption and waste, hence reducing cost. Moreover, due to their miniaturization and automation, they can be used as point-of-care devices, in areas that lack the infrastructure needed by conventional laboratories [3].

Due to these advantages, biochips are expected to revolutionize clinical diagnosis, especially immediate point-of-care diagnosis of diseases. For example, the

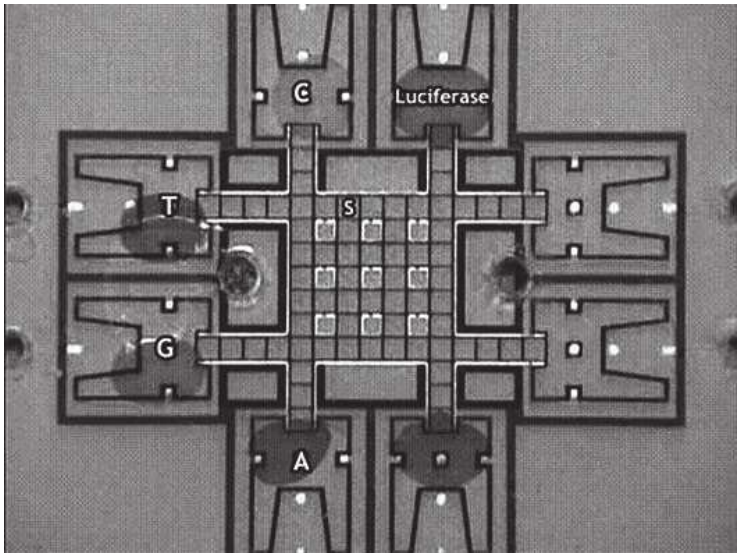


Figure 1.3: Advanced Liquid Logic's chip for DNA sequencing [13]

digital microfluidic biochip (DMB) shown in Figure 1.2 was proposed for the detection of Plasmodium parasites in human blood [8]. Such parasites are transmitted via infected mosquitoes and lead to malaria, one of the most common infectious diseases worldwide. The architecture of the designed chip is specific to malaria detection, with 13 reservoirs containing the samples and reagents, mixing and detection areas. A sequence of electrodes forming a bus are used for transporting the droplets through the different areas of the biochip during the assay. The advantages of such a miniaturized device compared to traditional microscopy-detection are portability, easiness of use and faster detection.

This is one example where a biochip can be used as a point-of-care device, with significant advantages over the standard diagnosis methodology. Another example is the biochip presented in [51], which is able to measure the level of glucose in human physiological fluids, and thus can be used by people suffering of diabetes. The device is based on a digital microfluidic biochip on which an optical detector consisting of a light emitting diode (LED) and a photodiode is integrated. After the glucose sample and the reagents are mixed on the microfluidic array the resulting droplet is brought to the optical detector, where the concentration of glucose is measured using the LED-photodiode setup. Due to the fast analysis time (less than 60 s) and the reduced amount of used reagents this biochip can be used successfully to replace the conventional measurement of glucose, using a spectrophotometer [51].

Moreover, biochips can be used for chemical detection of explosives in soil or water, in order to detect land mines and contamination of water with trinitrotoluene (TNT). Preliminary results were obtained in [41] for the detection of TNT, using a lab-on-a-chip device where the concentration of explosives was measured using colorimetric detection.

It has been also shown that digital microfluidic biochips can be successfully used in applications related to genetic engineering. Several biochips [65], [26], [42] have been proposed for performing polymerase chain reaction (PCR), a key technique in modern biology used for amplifying a piece of DNA. Biochips can also be used for determining the order of nucleotides in DNA (DNA sequencing), such an example being the chip developed by Advanced Liquid Logic and shown in Figure 1.3.

Other emerging application areas for biochips including drug discovery and tissue engineering [13]. Biochips can also be used in monitoring the quality of air and water, through real-time detection of toxins [13].

This chapter presents the motivation behind our research, a brief introduction to the considered problem and the contributions brought by this thesis. An overview of the thesis is given in the end of the chapter.

1.1 Motivation

Due to their advantages compared to traditional laboratories, biochips are expected to revolutionize many fields of biotechnology, such as clinical diagnosis, drug discovery, DNA analysis (e.g., polymerase chain reaction and nucleic acid sequence analysis), protein and enzyme analysis and immuno-assays [3].

Considering the potential of such devices for the biotechnology industry, the number of companies and research groups interested in biochips has increased substantially in recent years. The market for biochips is expected to increase, and reach US $3.4 billion by 2012, as stated by Global Industry Analysts, Inc. United States [17]. However, there are still challenges to be met in order for biochips to become widely commercialized. Most difficulties come from the complexity of these devices, which are the product of different energy domains (e.g., fluidic, electric, thermal). This mixture of domains implies that the design and test methods currently available for other devices (e.g., integrated circuits, micro-electromechanical systems) cannot be used directly for biochips, which exhibit unique characteristics and faults [4], [3]. Therefore, new methods are required, which consider the constraints specific to this new technology. The

complexity of biochips is expected to further increase, as the number of assays performed concurrently on the chip is becoming more and more significant.

In order to support the increase in biochip complexity and therefore their market growth, computer aided design (CAD) tools are required, which can offer the same level of support as the one taken for granted currently in the semiconductors industry. Initially, designers have used a bottom up approach for the design of biochips, combining fluidic components to create specific-application devices [11]. However, this bottom-up approach does not scale to the new designs. Consequently, top-down design methods have been proposed in [5], increasing the level of abstraction in biochip synthesis. Such techniques are necessary in order to improve the design of biochips, and to hide the implementation details of running biochemical assays from the users [3].

In this thesis we propose several top-down synthesis approaches for digital microfluidic biochips. Such techniques will reduce the design cost and improve productivity, and are the key to the further growth and market penetration of biochips [4].

1.2 Problem Formulation

A digital microfluidic biochip is composed of a two-dimensional array of electrodes, together with reservoirs for storing the liquids. Basic operations, such as mixing and dilution, are performed by routing the droplets on the microfluidic array.

Considering the architecture of digital microfluidic biochips, the design tasks that have to be performed during the synthesis problem have similarities to the high-level synthesis of very large-scale integration (VLSI) systems. Motivated by this similarity, researchers have started to propose approaches[1] for the top-down design of such biochips.

The following are the main design tasks that have been addressed:

- During the design of a digital microfluidic biochip, the bioassay protocols have to be mapped to the on-chip modules. The protocols are *modeled* using sequencing graph models, where each node is an operation, and each edge represents a dependency.

[1] See Chapter 4 for a presentation of related work.

- Once the protocol has been specified, the necessary modules for the implementation of the protocol operations will be selected from a module library. This is called the *allocation* step.
- As soon as the *binding* of operations to the allocated modules is decided, the *scheduling* step determines the time duration for each bioassay operation, subject to resource constraints and precedence constraints imposed by the protocol.
- Finally, the chip will be synthesized according to the constraints on the types of resources, cost, area and protocol completion times. During the chip synthesis, the *placement* of each module on the microfluidic array and the *routing* of droplets from one module to another have to be determined.
- All of the presented design tasks have to take into account possible defects during the fabrication of the microfluidic biochip. Thus, *testing* and *reconfiguration* have to be performed.

The synthesis problem for digital microfluidic biochips can be formulated as follows:

Given:

- a biochemical application containing the microfluidic operations to be performed and modeled using a sequencing graph;
- the design specifications for a biochip (dimensions of the microfluidic array, maximum number of reservoirs and optical detectors that can be integrated on the chip); and
- a library characterizing the completion time of operations,

we are interested to synthesize an implementation consisting of the allocation, binding, scheduling and placement such that the completion time of the biochemical application is minimized.

Application completion time minimization is particularly important for applications such as environmental monitoring and clinical diagnostics. Moreover, it reduces the effects that variations in the environment (e.g., temperature) can have on the integrity of the used samples and reagents [4]. In addition, completion time minimization allows us to use smaller areas and thus, reduce costs. Other objectives for the synthesis problem (not directly considered in this thesis) may include area minimization and fault tolerance maximization [4].

The synthesis problem and the corresponding design tasks will be explained in detail in Chapter 3.

1.3 Contributions

All of the previous work (see Chapter 4 on related work) considers that operations are performed on virtual devices (also called modules) of rectangular shape, which have a fixed placement on the microfluidic array. All electrodes of the device are considered occupied during the operation execution, although the droplet uses only one electrode at a time.

However, as will be discussed in Chapter 2), the operations can actually execute by routing the droplets on any sequence of electrodes on the microfluidic array.

- The main contribution of the thesis is a new model of operation execution (presented in Section 2.2.1), where we eliminate the concept of virtual modules. Thus, during the execution of the operation, we allow the droplets to move on the chip on any route (hence the name, routing-based operation execution, as opposed to module-based operation execution).
- The completion times of module-based operations is determined experimentally and stored in a library for each module dimension. However, for routing-based operation execution, the completion times depend on the route taken by the droplet. In Section 2.2.1 we have proposed an analytical method [31] for determining the percentage of operation completion for any given route. Our method provides safe estimates by decomposing the modules of a given module library determined experimentally.
- Using the routing-based model, in Chapter 8 we have proposed a routing-based synthesis approach based on a Greedy Randomized Adaptive Search Procedure (GRASP) [31]. We have shown that such a routing-based approach obtains the best application completion times across all the synthesis methods, since it can best utilize the available chip area. The disadvantage of the routing-based synthesis is that it can contaminate larger areas of the biochip (the potentially contaminating operation is not contained to a fixed rectangular area as in the case of module-based operation execution). However, we have shown in Section 8.3.2 how contamination during routing-based synthesis can be successfully addressed.
- The routing-based model is very flexible and has allowed us to explore other models of operation execution, in-between the two extremes: module-based (least flexible) and routing-based (most flexible). Thus, in Chapter 7

we have proposed a model where the routes are constrained inside a module (called droplet-aware module-based execution) [34]. By knowing at all times the positions of droplets inside a module we can relax the assumption that all electrodes of the module are occupied during operation execution, thus improving the utilization of the biochip area.

- The analytical method from Section 2.2.1 has allowed us to determine the completion times of operations on non-rectangular devices (e.g., "L" or "T" shapes). In Section 6.2 we have proposed a synthesis method using non-rectangular modules [32], where although all electrodes of the module are considered occupied throughout the execution, we can better utilize the available, but fragmented, space on the biochip, during the application execution.

- The synthesis approaches presented in Chapters 6 and 7 use a metaheuristic optimization called Tabu Search. In Chapter 5, we have proposed a Tabu Search algorithm [30] for the original module-based synthesis, which we have later extended in Chapter 6 and 7. We have shown that our Tabu Search-based approach is able to outperform all other synthesis approaches. Module-based synthesis assumes that modules are fixed during their execution. A natural extension, presented in Section 6.1, is to consider that modules can be moved during their execution to a new location (but keeping the same shape). We have shown that this approach reduces the biochip area fragmentation that occurs during the application execution [30].

- The original module-based synthesis approach is formally presented in Chapter 3, using an Integer Linear Programming (ILP) formulation [33]. The ILP formulation is useful in determining the optimal solutions and reasoning about the synthesis problem complexity. Although the ILP formulation can only handle small examples, we have used it to determine the quality of the proposed Tabu Search.

1.4 Thesis Overview

This thesis contains nine chapters organized as follows:

- **Chapter 2** introduces the architecture of digital microfluidic biochips, as well as their domains of applicability. The main types of biochemical operations are presented, and their execution on a DMB is discussed. We introduce a new, routing-based, model of operation execution and propose

an analytical method for determining the completion time of an operation on any given route. Finally, the graph model describing a biochemical application is introduced, and several examples of real-life applications are presented.

- **Chapter 3** contains a detailed formulation of the synthesis problem for digital microfluidic biochips, modeled using integer linear programming. The advantages of performing architectural level synthesis and placement in an unified manner are also discussed in this chapter.

- **Chapter 4** presents the related work in the area of synthesis approaches for digital microfluidic biochips.

- In **Chapter 5** we propose a Tabu Search-based algorithm for solving the synthesis problem introduced in Chapter 3. The method assumes that reconfigurable microfluidic operations are performed on virtual devices whose location and shape remain fixed throughout the execution of operations.

- In **Chapter 6** we modify the Tabu Search-based methodology to take better advantage of the dynamic reconfigurability characteristics of DMBs. Compared to the traditional operation execution (on fixed rectangular modules), we evaluate the improvements brought by moving a module and by changing the shape of the device on which an operation is bound during its execution.

- In **Chapter 7** we present a module-based synthesis approach with droplet-aware operation execution. We show that by considering the exact positions of droplets inside modules during operation execution we can better utilize the chip area and hence, significantly reduce the application completion time.

- In the routing-based operation execution, we eliminate the concept of virtual modules and allow the droplets to move on the chip on any route. In **Chapter 8** we propose a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm for routing-based synthesis. The algorithm is then extended to consider contamination avoidance during routing-based synthesis. This is particularly important for applications involving liquids that can contaminate the substrate on which they are transported.

- **Chapter 9** presents the conclusions of this thesis and discusses future ideas on how to extend the presented work.

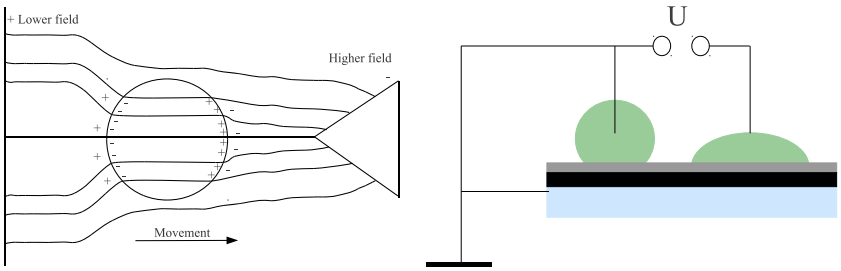
Chapter 2

Biochip Architecture and System Model

2.1 Biochip Architecture

A digital microfluidic biochip is typically composed of a microfluidic array of electrodes, together with reservoirs for storing the samples and reagents. The architecture of the array is dependent on the actuation mechanism used for creating and manipulating the droplets. The most used methods are dielectrophoresis (DEP) and electrowetting-on-dielectric (EWOD). Both methods are based on electrical forces and can provide high transportation speeds for droplets, using simple biochip architectures [5].

Dielectrophoresis [62] is a phenomenon that appears when a dielectric particle is subjected to a non-uniform electric field. In the absence of a field, a particle placed in vacuum contains dipoles that are randomly oriented in space. When a non-uniform electric field is applied, the particle becomes polarized due to the orientation of the dipoles parallel to the lines of the field. As a result a DEP force appears, transporting the particle in the direction of maximum or minimum electric field. The direction depends on the polarization of the droplet with respect to the medium. If the particle is more polarizable, the DEP force will transport it in the direction of the maximum field (positive



(a) Positive DEP (b) Change of contact angle in EWOD

Figure 2.1: Actuation methods for DMBs

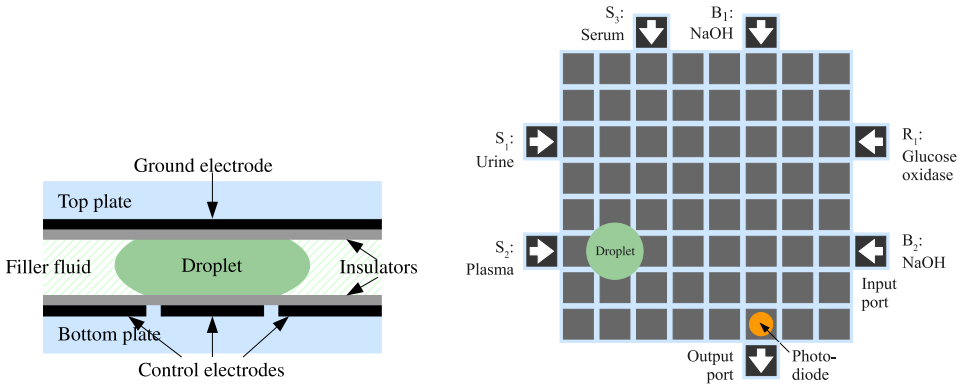
DEP, see Figure 2.1a). Otherwise, the particle is moved towards the minimum of the electric field (negative DEP). Due to the dependency of the movement on the particles' properties, DEP is successfully used in applications that require sorting particles with different characteristics [62].

In this thesis we consider digital microfluidic biochips based on the EWOD actuation method.

Electrowetting-on-dielectric [62] is based on surface tension, a property of liquids that becomes dominant in microfluidics, due to high surface-to-volume ratios. Surface tension is a result of unbalanced cohesive forces at the surface of a liquid. Because the surface molecules are subjected to cohesive forces only from the interior of the liquid, they will be more attracted to their neighbors and will assume a shape that has the least amount of surface area [62]. Let us consider Figure 2.1b. Initially the droplet is resting on an electrode, the two being separated by an insulator coated with a hydrophobic layer. Because the liquid is repelled by the hydrophobic molecules, the droplet does not spread out on the solid surface, assuming an almost spherical shape. Let us denote the initial contact angle between the droplet and the surface by θ_0. If a voltage V is applied between the liquid and the electrode, the contact angle θ changes according to the Lippmann-Young equation [43]:

$$cos\theta = cos\theta_0 + \frac{\epsilon_0 \epsilon_d}{2d\sigma_{LG}} V^2 \tag{2.1}$$

where ϵ_0 is the permittivity of the medium, ϵ_d and d are the dielectric constant and thickness of the insulating layer, respectively, and σ_{LG} is the surface tension between the liquid and the medium.

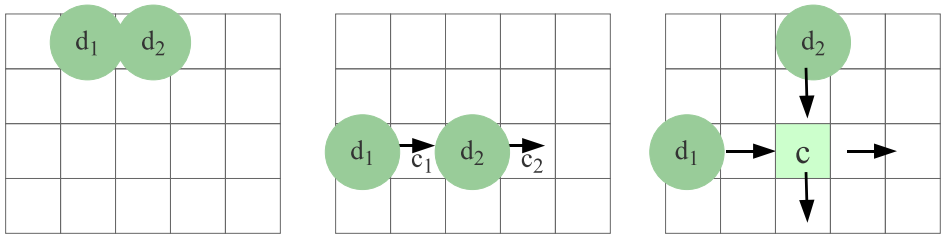


(a) Cell architecture (b) Biochip: array of cells

Figure 2.2: Biochip architecture

The change in the contact angle leads to a change in the wettability of the droplet, which returns to its hydrophobic state when the voltage is removed. If the voltage is applied to only one side of the droplet, the gradient in the contact angle at the two edges of the liquid will cause a surface stress in the direction of the applied voltage, leading to the movement of the droplet [43]. For example, turning off the middle control electrode and turning on the right control electrode in Figure 2.2a will force the droplet to move to the right. To avoid the unexpected mixing of liquids, fluidic constraints must be enforced, ensuring that a minimum distance is kept between droplets executing on the microfluidic array. Consider for example droplets d_1 and d_2 in Figure 2.3a. If the two droplets are situated on adjacent electrodes (as shown in the figure), they will tend to merge and form a single large droplet. If merging of the liquids is to be avoided, a spacing of at least one cell must be kept between the two droplets, at any time. A similar situation is presented in Figure 2.3b, where droplets d_1 and d_2 are to be transported in the directions shown by the arrows. Let us consider that the electrodes denoted in the figure by c_1 and c_2 are activated. Since droplet d_2 has two adjacent activated electrodes (c_1 and c_2) it will not move significantly, resulting in a merging with droplet d_1. In order to avoid such a situation, there must be only one activated neighboring electrode for each droplet on the microfluidic array.

A DMB is typically composed of an array of electrodes, together with reservoirs for storing the samples and reagents. The architecture of the biochip proposed in Figure 1.2 is optimized for the protocol specific to malaria detection, containing a general bus used for transporting droplets between the different components of the chip (e.g., reservoirs, mixer, detector). However, as biochips are expected



(a) Static fluidic constraint (b) Dynamic fluidic constraint (c) Droplet contamination

Figure 2.3: Examples of droplet contamination

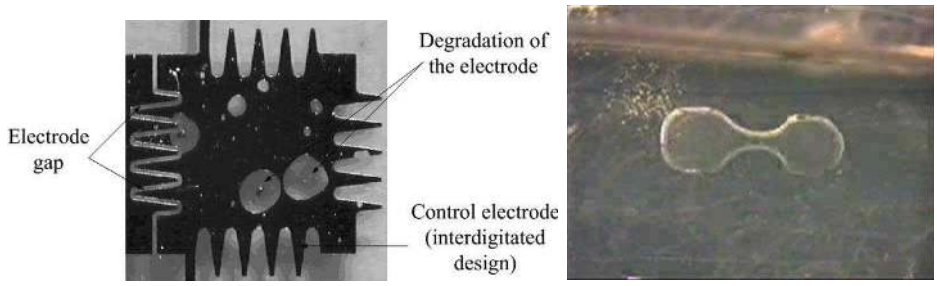
to perform thousands of assays in parallel, their architecture will become less application-specific. Throughout this thesis we assume a general architecture of a DMB, in which operations are performed on a two-dimensional array of electrodes.

The schematic of this general architecture is presented in Figure 2.2b. The chip is composed of a microfluidic array of identical cells, together with reservoirs for storing the liquid. Each cell is composed of two parallel glass plates, as shown in Figure 2.2a. The top plate contains a single indium tin oxide (ITO) ground electrode, while the bottom plate has several ITO control electrodes. The electrodes are insulated from the droplet through an insulation layer of ParyleneC, on which a thin film of Teflon-AF is added [50]. The role of the Teflon layer is to provide a hydrophobic surface on which the droplet will move. The two parallel plates are separated through a spacer, providing a fixed gap height. The droplet moves between the two plates, in a filler fluid (e.g., silicone oil), used in order to prevent evaporation and the adhesion of molecules on the surface of the chip [4].

Fabrication of digital microfluidic biochips

In order to decrease the cost of biochips, printed circuit board (PCB) technology has been proposed recently as a substrate for building inexpensive biochips. A typical PCB chip is built using copper layer for electrodes, solder mask as the insulator and Teflon AF as the hydrophobic layer. Such devices can be fabricated using existing PCB techniques and require higher operation voltages than glass substrates, due to their rougher surfaces [19]. The main advantage of PCB chips consists in the reduced fabrication costs, which makes them ideal for applications which require disposable devices.

Initial designs have considered that each electrode is controlled individually, using a dedicated pin. This direct-addressing scheme is still used successfully



(a) Insulator degradation [61] (b) Imperfect splitting [12]

Figure 2.4: Examples of faults for DMBs

for small to medium-size electrode arrays. However, for large-size arrays ($>$ 10×10 electrodes), the increase in the number of required pins leads to higher wiring complexity and thus to higher costs [23]. Thus, several pin-addressing schemes have been proposed recently for reducing the number of control pins. For example, the cross-referencing scheme proposed in [48] allows the control of a $n \times m$ microfluidic array using only $n + m$ pins. A biochip using this scheme is required to have electrode rows placed orthogonally on both glass plates, a droplet being moved by activating either the top or the bottom row of electrodes, depending on the direction of movement [48]. The disadvantage of this scheme is the fact that it requires a special structure of the chip, with electrode rows on both plates. Another scheme, proposed in [72] reduces the number of control pins by connecting the ones with "compatible" actuation sequence. The compatibility of pins is decided by analyzing the scheduling and routing information for performing an application on the chip.

The details regarding droplet movement are stored in a microcontroller, which coordinates the activation of the electrodes on the microfluidic array.

In this thesis we do not focus on pin-count reduction and hence we consider direct-addressing biochips, in which each electrode is controlled individually.

Faults specific to digital microfluidic biochips

As biochips are expected to be used for safety-critical applications, it is important that the faults that they can exhibit are well known and that the proper actions are taken in order for the devices to function properly. There are two types of faults that can appear in a digital microfluidic biochip [59]: catastrophic and parametric.

Catastrophic faults are generally caused by physical defects and lead to the complete malfunctioning of part of the biochip. Some examples of causes leading to catastrophic faults are given as follows [59]:

- Dielectric breakdown — occurs when a high voltage applied to an electrode produces the breakdown of the dielectric, creating a short between the electrode and the droplet. As a result, the movement of the droplet resting on the corresponding electrode is affected.
- Degradation of the insulator (see Figure 2.4a)— happens gradually, during the operation of the biochip. When the degradation level reaches a certain threshold, the movement of the droplet from the corresponding electrode is affected.
- Short between two adjacent electrodes — leads to the formation of one large electrode, occupying the surface of the two electrodes. As the surface of the newly create electrode is too big, the droplet resting on it is not large enough to overlap with the adjacent electrodes. As a result, the droplet can no longer be transported.

Parametric faults do not result in the malfunctioning of the biochip. Rather, they affect the performance of the system. Examples of parametric faults include [59]:

- Increased viscosity of the fluid filler — can result in erroneous concentrations in the case of mixing operations.
- Electrode contamination — caused by certain substances (e.g., proteins, peptides) that tend to adsorb on the electrodes they are routed on. This can lead to the contamination of the droplets that are transported on the same surface, at a later time.
- Imperfect splitting — can be caused by the misalignment of the droplet with the control electrode, during a split operation. Consequently, the volumes of the two resulted droplets are unbalanced, as shown in Figure 2.4b. Such variability in droplet volume can propagate throughout the execution of the application, resulting in an erroneous output of the bioassay.

In this thesis we do not consider catastrophic faults that can appear during the operation of a digital microfluidic biochips. Several methods for the detection of catastrophic faults have been proposed by researchers, and can be used for

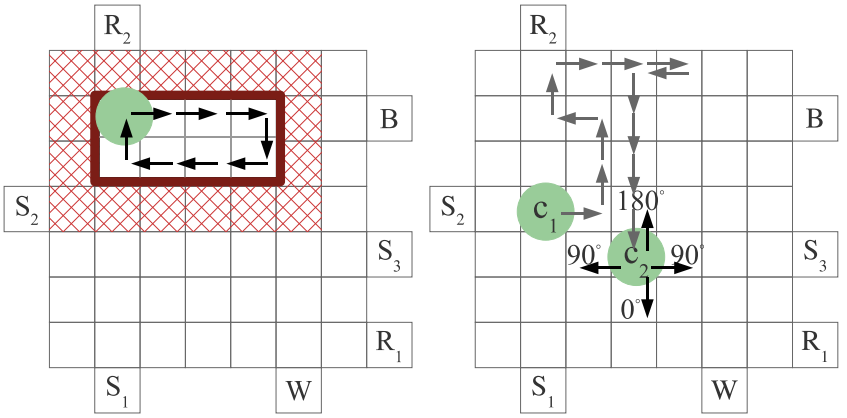
ensuring the correct functioning of the microfluidic platform [60],[58]. However, in Chapter 8 we propose a method for droplet contamination avoidance during the synthesis of DMBs. This is a parametric fault that can appear during the execution of a biochemical application, if the purity of a droplet is accidentally affected.

In order for the outcome of a biochemical assay to be reliable it is important that the samples and reagents are not accidentally contaminated throughout the execution of operations. The outcome of a bioassay can be influenced by modifications in the hydrophobic surface of the chip. Certain molecules used in biochemical applications (e.g., lipids, proteins, DNA, peptides) can adsorb onto hydrophobic surfaces, fouling them. Adsorption of such molecules must be avoided, as it can contaminate the other liquids present on the microfluidic array and it can even affect the actuation process [37]. Let us consider droplets d_1 and d_2 in Figure 2.3c, which must be routed on the chip in the directions shown by the arrows. We assume that droplet d_1 contains protein molecules and is routed first. As a result, molecules will adsorb to the device surface, leaving traces on the cells on which droplet d_1 is routed. Therefore, when droplet d_2 is transported over the cell denoted by c it is contaminated by the protein traces left behind by d_1. Such changes in the purity of samples and reagents must be avoided, as they can affect the correct functioning of the whole assay. One method of reducing surface fouling is by transporting droplets in an immiscible medium (e.g., silicone or fluorinated oil), such that the fluids are not in direct contact with the hydrophobic surface [14], [13]. When the immiscible liquid can not completely avoid contamination, wash droplets are used for cleaning the device surface during the execution of the bioassay [38].

2.2 Operation Execution: Module vs. Routing

Using the architecture in Figure 2.2, and changing correspondingly the control voltages, all of the required operations, such as transport, splitting, dispensing, mixing, and detection, can be performed.

For example, mixing is done by bringing two droplets to the same location and merging them, followed by the transport of the resulted droplet over a series of electrodes. By moving the droplet, external energy is introduced, creating complex flow patterns (due to the formation of multilaminates), thus leading to a faster mixing [40]. Mixing through diffusion, where the resulted droplet remains on the same electrode, is very slow. The operation can be executed anywhere on the microfluidic array and is not confined to a certain area, thus we say that mixing is a "reconfigurable" operation. Another reconfigurable operation



(a) Module-based operation (b) Routing-based operation

Figure 2.5: Execution of a mixing operation

is dilution, which consists of a sequence of mixing and splitting steps [46]. A biochemical application may also contain "non-reconfigurable" operations, that are executed on real devices, such as reservoirs or optical detectors.

So far, researchers have considered that reconfigurable operations are performed inside virtual modules, created by grouping adjacent cells. Such a module is shown in Figure 2.5a, where the droplet is routed circularly on a series of electrodes until the mixing operation is completed. The trajectory of the droplet inside the module is described by a movement pattern, represented by the arrows inside the virtual module.

Table 2.1 presents the results of the experiments performed in [40], where several mixing times were obtained for various areas, creating a module library. One problem addressed by the experiments is flow reversibility, when complex patterns inside the droplet are unfold into simpler ones when the direction in

Operation	Area (cells)	Time (s)
Mixing	2 × 4	2.9
Mixing	1 × 4	4.6
Mixing	2 × 3	6.1
Mixing	2 × 2	9.95
Dispensing	–	2
Detection	1 × 1	30

Table 2.1: Module library

which the droplet is transported is changed by 180°. This is the case of linear mixers (e.g., Figure 2.6b), where the motion of the droplet is bidirectional. One solution to avoid flow reversibility is to transport the droplet in a circular motion, as in the 2 × 2 virtual module shown in Figure 2.6d. However, it has been shown that since the droplet is rotating around the pivot point in the center of the created module, part of the droplet remains unmixed and thus the operation takes longer (9.95 s) to complete. In the 2 × 3 module shown in Figure 2.6c two additional electrodes are introduced to eliminate the static pivot point present in the 2 × 2 module, thus reducing the mixing time to 6.1 s. The mixing time is further improved for the 2 × 4 mixer in Figure 2.6a, leading to a 2.9 s completion time. The experiments show that faster mixing is obtained by moving the droplet linearly for as long as possible, reducing flow reversibility.

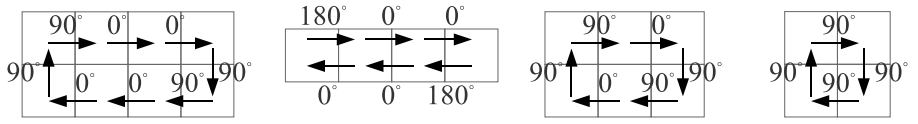
During module-based operation execution, all cells inside the module are considered occupied, although the droplet uses only one cell at a time. Thus, the remaining cells cannot be used for other operations, which is inefficient since it reduces the potential for parallelism. In addition, in order to prevent the accidental merging of a droplet with another droplet in its vicinity, a minimum distance must be kept between operations executing on the microfluidic array. For example, in Figure 2.5a these fluidic constraints are enforced by surrounding the module by a 1-cell segregation area (the hashed area), containing cells that can not be used by other operations until mixing finishes.

An alternative to modules, proposed in this thesis, is routing-based operation execution. As mixing is performed by routing, an operation can be executed anywhere on the array, unconstrained by a rectangular shape representing a virtual module. This characteristic of the mixing operation is shown in Fig. 2.5b, where the droplet is routed freely on a sequence of electrodes, according to the shown route.

2.2.1 Characterizing Routing-Based Operation Execution

A contribution of this thesis is the characterization of routing-based operation execution. This characterization is used throughout the thesis to determine the operation completion time for: modules with non-rectangular shape (Chapter 6), modules with droplet-aware operation execution (Chapter 7) and operations executing on any sequence of electrodes on the microfluidic array (Chapter 8).

We propose an analytical method for determining how the percentage of operation execution varies depending on the movement of the droplet. Our method provides safe estimates by decomposing the devices from Table 2.1.



(a) 2 × 4 module (b) 1 × 4 module (c) 2 × 3 module (d) 2 × 2 module

Figure 2.6: Characterization of module library

Let us consider that while mixing the droplet in Figure 2.5b reaches the cell c_2 at time t. The previous movements for the droplet are as shown by the arrows. We have five possibilities for $t + 1$: routing the droplet to the left, to the right, up, down or keeping the droplet on c_2. Let us denote with p^0 the percentage of mixing obtained while routing the droplet on an electrode in a forward movement (relative to the previous move), with p^{90} the percentage obtained from a perpendicular movement of the droplet and with p^{180} the percentage of mixing obtained from a backward movement, see Figure 2.5b.

Considering Table 2.1, we can estimate the percentage of mixing over one cell, corresponding to each type of movement (forward, backward, perpendicular). In this thesis we consider the data[1] from [43], which allows us to approximate that the time required to route the droplet one cell is 0.01 s. In order to approximate p^0, p^{90} and p^{180} we decompose the movement patterns from the module library in Table 2.1 in a sequence of forward, backward and perpendicular motions, as shown in Figure 2.6. For example, the 2 × 2 mixer in Figure 2.6d can be decomposed in perpendicular movements, because after each move the droplet changes its routing direction by 90°. As shown in Table 2.1, the operation takes 9.95 s to execute inside the 2 × 2 module, thus we can safely approximate the percentage of mixing p^{90} to 0.1%.

For the 2 × 3 module shown in Figure 2.6c, the movement pattern is composed of forward and perpendicular movements. By considering the mixing time shown in Table 2.1 and $p^{90} = 0.1\%$, we obtain the percentage of mixing resulted from one forward movement $p^0 = 0.29\%$. Note that by decomposing the 2 × 4 module shown in Figure 2.6a, we obtain a different value for p^0: 0.58%. This is because the forward mixing percentage is not constant, but it depends on the number of electrodes used. Therefore we consider that there are two values that estimate the percentage of forward movement: p_1^0, when the forward movement is continued only for one cell as in Figure 2.6c, and p_2^0, when the forward movement of the droplet is of at least two cells. This is a pessimistic approximation, since the value of p^0 will further increase if the droplet continues to move forward.

[1]Electrode pitch size = 1.5 mm, gap spacing = 0.3 mm, average linear velocity = 20 cm/s.

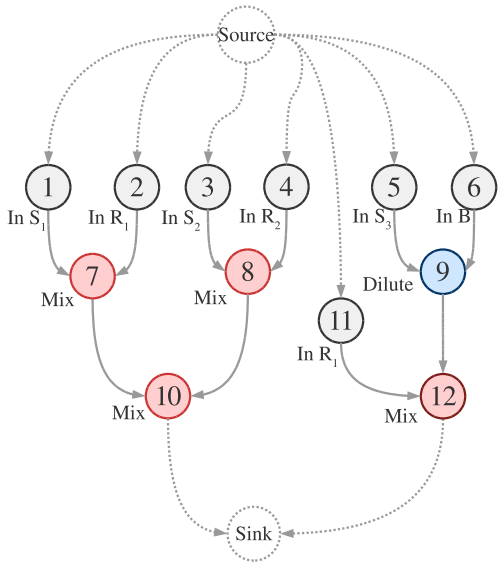


Figure 2.7: Application graph

Considering the percentage of forward movement p_2^0 in the decomposition of the 1×4 module in Figure 2.6b, we obtain the pessimistic percentage of mixing performed during a backward motion: $p^{180} = -0.5\%$. The negative mixing is explained by the unfolding of patterns inside the droplet, i.e., the two droplets tend to separate when moved backward.

Using these percentages, we can determine the operation completion time for any given route. For example, in Figure 2.5b, we have 3.19% of the mixing completed in 0.13 s.

2.3 Application Model

In order to perform a biochemical application on a biochip, its protocol must be known, that is the sequence of basic operations (e.g., dispensing, mixing, dilution, detection) composing the application. We assume that such protocols will be provided by the users of the biochips, e.g., biochemists.

The protocol of a biochemical application can be modeled using an abstract model consisting of a sequencing graph [6]. The graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is directed, acyclic and polar (i.e., there is a *source node*, which is a node that has no predecessors

and a *sink node* that has no successors). Each node $O_i \in \mathcal{V}$ represents one basic microfluidic operation.

An edge $e_{i,j} \in \mathcal{E}$ from O_i to O_j indicates that the output of operation O_i is the input of O_j. An operation can be activated after all its inputs have arrived and it issues its outputs when it terminates.

For example in Figure 2.7 we have an example of an application graph with twelve operations, O_1 to O_{12}. The application consists of four mixing operations (O_7, O_8, O_{10} and O_{12}), one diluting operation (O_9) and six input operations (O_1, O_2, O_3, O_4, O_5, O_6 and O_{11}).

In order to perform the operations, the required volumes for the corresponding droplets must be known. In microfluidic biochips, these volumes are in the range of nano- or microliters. As the actuation of liquids depends on the parameters of the biochip (e.g., electrode pitch, gap spacing), it must be ensured that the volume of a droplet allows the liquid to be moved. For example, when two droplets are mixed, it is likely that the resulted droplet is too large to be transported on the microfluidic array. In such cases split operations must be inserted, to adjust the volume of the droplet to an acceptable value. In this thesis we assume that each mixing operation is followed by a split, in order to maintain the droplet volume. The split operations are not explicitly shown in the application graphs used throughout the thesis, however, we assume that the time required for a split is included in the time for mixing execution.

The details of the execution of the operations on a biochip are the result of a synthesis process, which decides when and where each operation is performed.

2.4 Case Studies

This section describes the modeling of three real-life biochemical assays, using application graphs.

2.4.1 Mixing Stage of the Polymerase Chain Reaction

Figure 2.8 describes the mixing stage of the polymerase chain reaction (PCR/M), a technique used for DNA analysis. In PCR, several thermal cycles are used to replicate a piece of DNA, creating thousands of copies. This method is particularly important when the quantity of existent material is too scarce in order to

successfully analyze the initial DNA sample. The first step of the polymerase chain reaction consists of seven mixing operations, denoted in Figure 2.8 by O_1 to O_7. The product resulted from this stage undergoes a series of temperature-cycles, required for DNA amplification [27].

2.4.2 In-Vitro Diagnostics on Physiological Fluids

Figure 2.9 describes the protocol for an in-vitro diagnostics assay (IVD) in which the level of different metabolites in human physiological fluids are measured. The graph contains input operations for the samples (urine, plasma, and serum), reagents (glucose oxidase, lactate oxidase) and buffer substance. The level of glucose and oxidase are measured for each type of physiological fluid, using detection operations.

2.4.3 Colorimetric Protein Assay

The application graph in Figure 2.10 describes a protein assay, a procedure used for determining the concentration of a certain protein in a solution. The protocol is based on a reaction between the protein of interest and a dye. The concentration of the protein is determined by measuring the absorbance of a particular wavelength in the resulted substance.

The protocol consists of 103 microfluidic operations and uses three types of liquids: physiological fluid (sample containing the protein), Coomassie Brilliant Blue G-250 dye as reagent and NaOH as buffer substance. Before being mixed with the dye, the sample is first diluted with the NaOH buffer using the mixing-splitting scheme proposed in [46]. The protocol finishes with detection operations, in which the protein concentration for the resultant solution is measured [56].

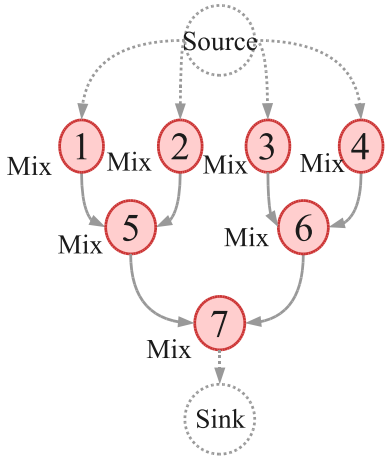


Figure 2.8: Mixing Stage of the Polymerase Chain Reaction Assay

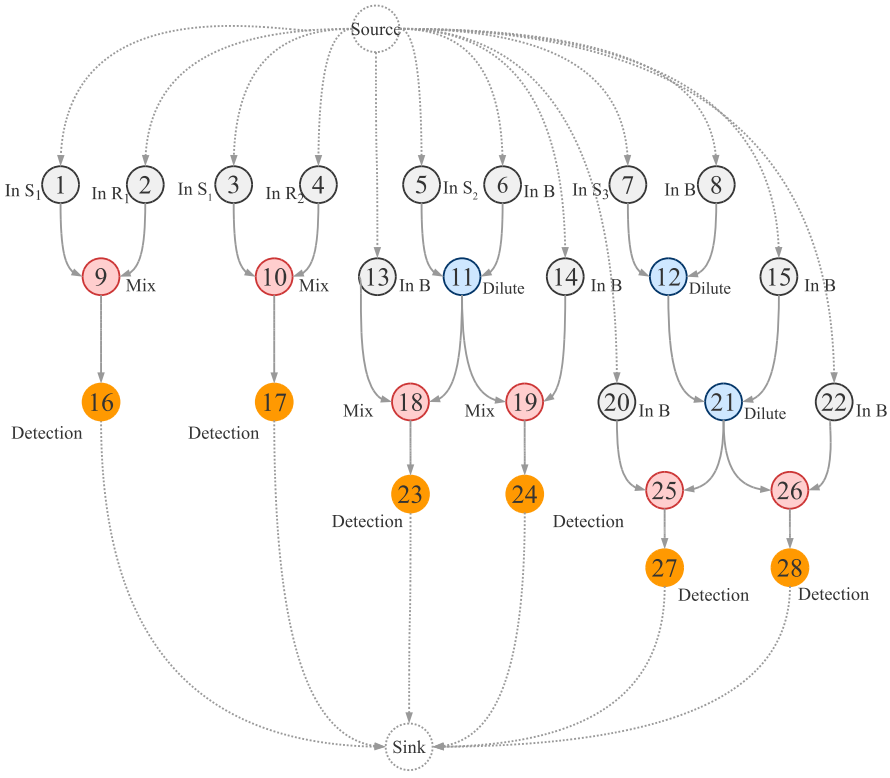


Figure 2.9: In-Vitro Diagnostics on Physiological Fluids

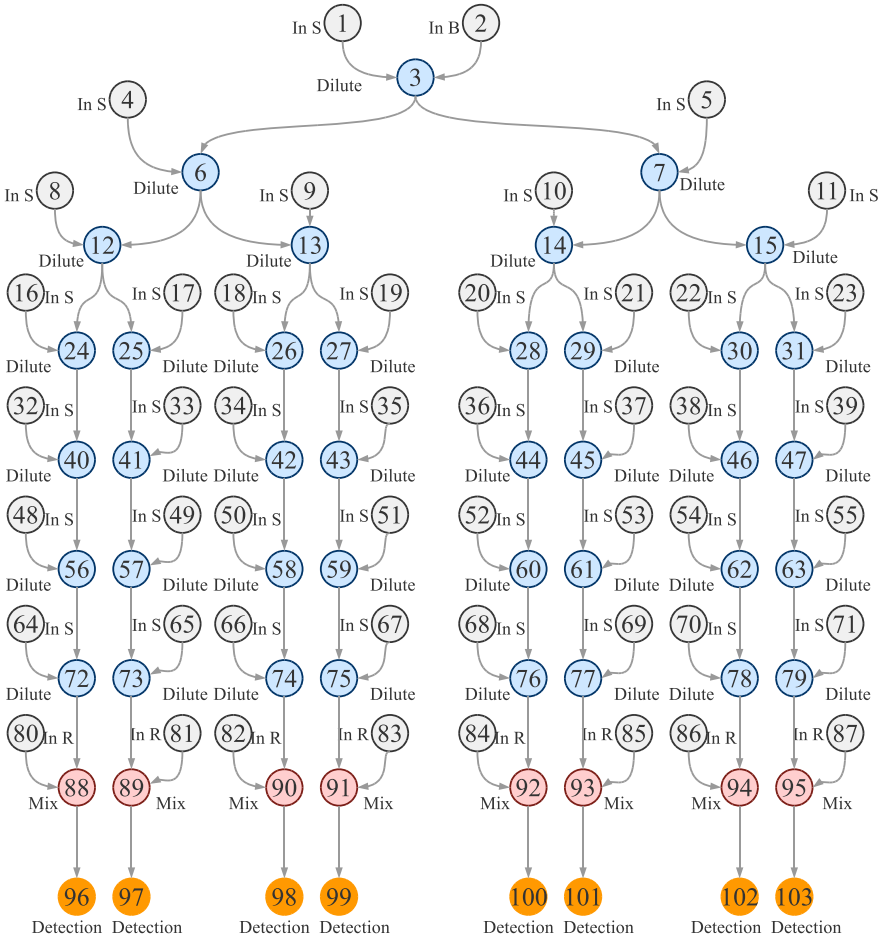


Figure 2.10: Colorimetric Protein Assay

Chapter 3

Synthesis Problem and ILP Formulation

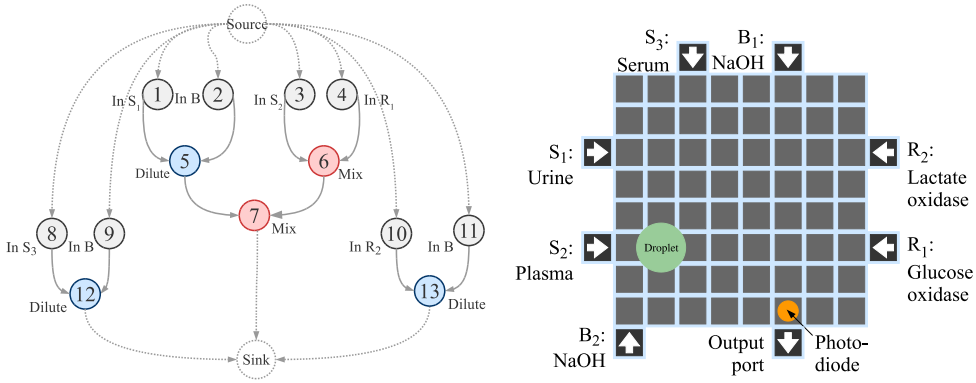
This chapter presents the synthesis problem of digital microfluidic biochips in the case when operations are performed inside rectangular virtual devices, which have a fixed position throughout their execution. An ILP formulation of the problem is given and the advantages of performing "unified" architectural-level synthesis and placement are discussed.

3.1 Synthesis Problem

The module-based synthesis problem can be formulated as follows. Given:

- a biochemical application modeled as a graph $\mathcal{G}$;
- a biochip consisting of a two-dimensional $m \times n$ array $\mathcal{C}$ of cells ; and
- a characterized module library $\mathcal{L}$,

we are interested to synthesize that implementation Ψ, which minimizes the schedule length $\delta_{\mathcal{G}}$ (i.e., the completion time of the application, t_{sink}^{finish}). Synthesizing an implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P}, \mathcal{R} >$, means deciding on:



(a) Application graph (b) Biochip

Figure 3.1: Synthesis example

- the allocation $\mathcal{A}$, which determines what modules from the library $\mathcal{L}$ should be used;
- the binding $\mathcal{B}$ of each operation $O_i \in \mathcal{V}$ to a module $M_k \in \mathcal{A}$;
- the schedule $\mathcal{S}$ of the operations, which contains the start time t_i^{start} of each operation O_i on its corresponding module; and
- the placement $\mathcal{P}$ containing the locations at which operations will be executed on the $m \times n$ array.

Routing determines the routes $\mathcal{R}$ taken by the droplets between modules and between modules and input/output ports.

Since routing times are one order of magnitude smaller than operation times (see the data in Section 2.2.1), routing has been considered so far as a post-synthesis step following the allocation, binding, scheduling and placement of modules on the microfluidic array.

In this chapter we focus on the architectural-level synthesis and placement for digital microfluidic biochips. Similar to the previous approaches, we assume that the routing of droplets will be determined during a separate step, following the placement of devices on the array.

Let us use the graph shown in Figure 3.1a to explain the architectural-level synthesis and placement for DMBs. We would like to implement the operations on the 8×8 biochip from Figure 3.1b. We consider the current time step as being t. We assume that a diluting operation from another application has been

scheduled at an earlier time step on module $Diluter_1$, has been placed on the microfluidic array as shown[1] in Figure 3.2c and will finish executing at $t + 4$.

The next subsections will illustrate the design tasks needed for synthesizing the application on the chip. The presentation order does not necessarily correspond to the order in which a synthesis approach performs these task.

3.1.1 Allocation and placement

The graph shown in Figure 3.1a contains two types of operations: non-reconfigurable (input) and reconfigurable (mixing and dilution). The scheduling of input operations is determined at the same time with the other operations, the fixed number of reservoirs representing a constraint to the final completion time of the application. However, as they execute outside the microfluidic array and do not affect the placement of the other operations, for simplicity reasons we ignore inputs in this example. We assume that the locations of reservoirs have been decided during the fabrication of the chip and are as shown in Figure 3.1b. We need to assign each input operation to a reservoir of the same type, e.g., O_2 can only be assigned to one of the buffer reservoirs B_1 and B_2. Let us consider that the input operations are assigned to the input ports as follows: O_1 to S_1, O_2 to B_1, O_3 to S_2, O_4 to R_1, O_8 to S_3, O_9 to B_1, O_{10} to R_2 and O_{11} to B_2. The synthesis approach will have to decide the scheduling of the input operations and make sure that each reservoir is used by at most one input operation at each time step.

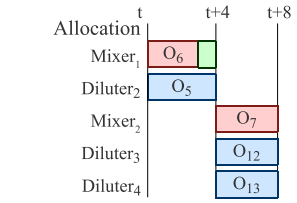
For the other operations in Figure 3.1a, the mixing operations (O_6 and O_7) and the dilution operations (O_5, O_{12} and O_{13}) our synthesis approach will have to allocate the appropriate modules, bind operations to them and perform the placement and scheduling.

Let us assume that the available module library is the one captured by Table 3.1. We have to select modules from the library while trying to minimize

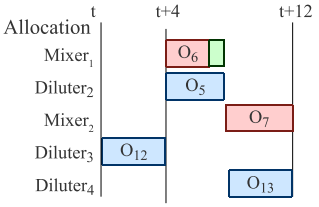
[1] In the figures we denote $Mixer_i$ with M_i and $Diluter_i$ with D_i.

Operation	Area (cells)	Time (s)
Mixing	2 × 4	3
Mixing	2 × 2	4
Dilution	2 × 4	4
Dilution	2 × 2	5

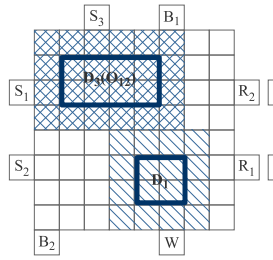
Table 3.1: Module library



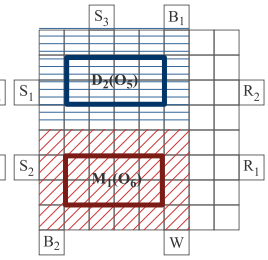
(a) Schedule under area constraints



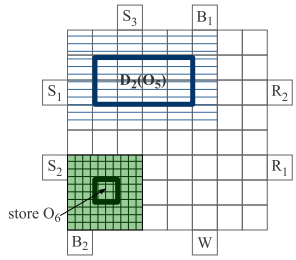
(b) Schedule with placement



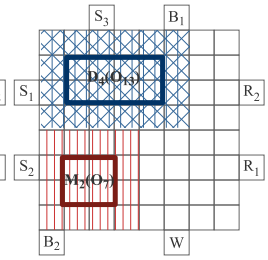
(c) Placement at t



(d) $t + 4$



(e) $t + 7$



(f) $t + 8$

Figure 3.2: Implementation example

the application completion time and place them on the 8×8 chip. We ignore the position of droplets inside modules, and we wrap devices with segregation cells, as explained in Chapter 2.

A simplification to the problem consists in separating the architectural-level and physical-level steps during the synthesis process. This is the approach taken in [53]. In this case, the allocation is only constrained by the chip area, $8 \times 8 = 64$ in our example. For our example, separating the two steps will lead to the allocation in Figure 3.2a (see the list of devices to the left of the schedule), where the following modules are used: one 2×2 mixer (4×4 with segregation area), one 2×4 mixer (4×6 with segregation area) and three 2×4 diluters (4×6 with segregation area). This allocation leads to the shortest completion time of the application, given the module library and the area constraints. Since $4 \times 4 + 4 \times 6 + 4 \times 6 = 64$, the architectural synthesis will wrongly assume that $Diluter_2$ and $Mixer_1$ can be placed concurrently on the array, at the same time with $Diluter_1$, which has already been placed at a previous time step. The resulted schedule is shown in Figure 3.2a. However, $Diluter_2$ and $Mixer_1$ cannot be placed on the chip at the same time with $Diluter_1$ without overlapping, and thus, during the placement step, the schedule will have to be modified. This will lead to the schedule in Figure 3.2b, where O_5 and O_6 will

be delayed until $t + 4$, when there is enough space on the microfluidic array to accommodate both $Diluter_2$ and $Mixer_1$. The schedule is depicted as a Gantt chart, where, for each module, we represent the operations as rectangles with their length corresponding to the duration of that operation on the module. The placement for this allocation is as indicated in Figure 3.2c–f.

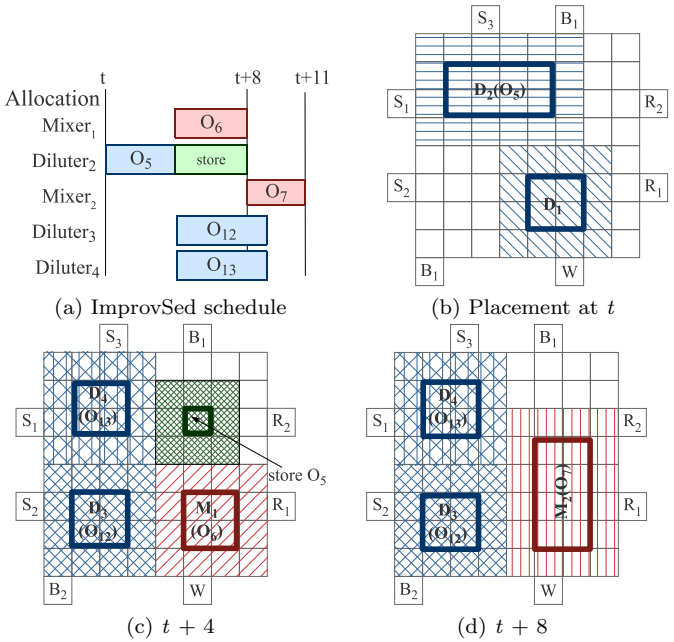
The placement problem of DMBs can also include finding the location of non-reconfigurable devices (e.g., reservoirs, optical detectors), whose number is constrained by the design specifications. As input operations are executed outside the microfluidic array, the positions of reservoirs can be determined manually, after the placement of the other devices. The locations of optical detectors on the array are decided during the placement step of the synthesis process and remain fixed throughout the execution of the application. If the synthesis process decides the mapping of a biochemical application to an already fabricated biochip, then the locations of non-reconfigurable devices are given as part of the input specifications.

3.1.2 Binding and Scheduling

Once the modules have been allocated and placed on the microfluidic array, we have to decide on which modules to execute the operations (binding) and in which order (scheduling), such that the application completion time is minimized.

Considering the graph in Figure 3.1a with the allocation presented in Figure 3.2a, Figure 3.2b presents the optimal schedule for the case when the placement is not considered during the architectural-level synthesis. For example, operation O_7 is bound to module $Mixer_2$, starts immediately after the diluting operation O_5 (i.e, $t_7^{start} = t + 8$) and takes 4 s, finishing at time $t_7^{finish} = t + 12$. We consider that the schedule is divided in time steps of one second, and we capture the set of time steps with $\mathcal{T}$.

Note that special "store" modules have to be allocated if a droplet has to wait before being processed. In general, if there exists an edge $e_{i,j}$ from O_i to O_j such that O_j is not immediately scheduled after O_i (i.e., there is a delay between the finishing time of O_i and the start time of O_j) then we will have to allocate a storage cell for $e_{i,j}$. Hence, the allocation of storage cells depends on how the schedule is constructed. In Figure 3.2b the droplet resulted from the mixing operation O_6 has to be stored until the dilution operation O_7 finishes executing. The placement in Figure 3.2e shows the 1×1 (3×3 with segregation cells) storage module. Any available cell on the microfluidic array can be used for temporarily storing the droplet.



(a) ImprovSed schedule (b) Placement at t

(c) $t + 4$ (d) $t + 8$

Figure 3.3: Unified allocation and placement

3.1.3 Unified Architectural-Level Synthesis and Placement

The schedule in Figure 3.2b can be improved if we consider placement at the same time with architectural-level synthesis, and not as a separate phase. This will lead to the allocation in Figure 3.3a, where the following modules are used: one 2×2 mixer (4×4 with segregation area), one 2×4 mixer (4×6 with segregation area), one 2×4 diluter (4×6 with segregation area) and two 2×2 diluters (4×4 with segregation area). As we can see from Figure 3.3a, considering a unified architectural-synthesis and placement approach leads to an improvement in the schedule length of the application, $t + 11$ s, as compared to $t + 12$ s in Figure 3.2b.

The next section presents an ILP formulation of the unified architectural-level and placement problem for digital microfluidic biochips.

3.2 ILP Formulation

In an ILP model a system is described by a minimization objective and a set of constraints which define valid conditions for the system variables. A solution to the modeled problem is an enumeration of all system variables, such that the constraints are satisfied. The optimization objective is specified as minimizing the completion time of the application,

$$\text{minimize } t_{sink}^{finish}, \tag{3.1}$$

where t_{sink}^{finish} is the finishing time of the sink node of the application graph.

The module library $\mathcal{L}$ is defined as a set of modules, each having a type (e.g., mix, dilute, store) and different characteristics in terms of area and execution time. The binding of operations to modules in the architecture is captured by the function $\mathcal{B} : \mathcal{V} \rightarrow \mathcal{A}$, where $\mathcal{A}$ is the list of allocated modules from the given library $\mathcal{L}$. We denote the execution time of an operation O_i on module $M_k = \mathcal{B}(O_i)$ where it is assigned for execution, by $C_i^{M_k}$.

During the iterations performed for finding the optimal solution, the ILP will bind each operation that needs to be scheduled to a module of the same type. As there can be more than one module of the same type defined in the library (e.g., a 2×2 mixer, a 2×4 mixer), the same operation can be bound to different modules during different ILP iterations.

Let us denote by $\mathcal{V}^+$ the set of all the operations to be performed during the execution of the application. Then $\mathcal{V}^+$ needs to contain not only the operations $O_i \in \mathcal{V}$, but also additional operations of type storage. Let us denote by $\mathcal{S}t$ the set of storage operations. We associate one storage to each operation in the graph. The rest of the operations in $\mathcal{V}^+$ are divided into two sets: reconfigurable $\mathcal{V}^{reconf}$ and non-reconfigurable operations $\mathcal{V}^{nonReconf}$.

The constraints fall under the following categories: i) scheduling and precedence; ii) resource constraints; iii) placement constraints. In order to be able to express them, a binary variable is defined as follows:

$$z_{i,j,k,l} = \begin{cases} 1, & \text{if operation } O_i \text{ starts executing at} \\ & \text{time step } j \text{ on module } M_k \text{ placed} \\ & \text{with its bottom-left corner over cell } c_l \\ 0, & \text{otherwise} \end{cases}$$

Such a variable captures the allocation and binding (operation O_i is executing on module M_k), the scheduling (O_i starts to execute at time step j, with a

duration of $C_i^{M_k}$) and the placement (the bottom-left corner of module M_k is placed over cell c_l).

By using the defined variable, the start time of an operation $O_i \in \mathcal{V}^+$ becomes:

$$t_i^{start} = \sum_j \sum_k \sum_l j \times z_{i,j,k,l} \quad \forall O_i \in \mathcal{V}^+, \tag{3.2}$$

where j represents the time step when the operation starts executing.

3.2.1 Scheduling and Precedence Constraints

The scheduling constraint requires that an operation O_i be scheduled only once:

$$\sum_j \sum_k \sum_l z_{i,j,k,l} = 1, \quad \forall O_i \in \mathcal{V}^+. \tag{3.3}$$

For each edge in the application graph we have to introduce a precedence constraint. Consider the operations O_i and $O_n \in \mathcal{V}$ for which there exists a dependency $e_{i,n} \in \mathcal{E}$ in the sequencing graph $\mathcal{G}$. Then O_n must be scheduled for execution only after the completion of O_i:

$$t_i^{start} + \sum_j \sum_k \sum_l \left(C_i^{M_k} \times z_{i,j,k,l} \right) \leq t_n^{start}, \forall O_i \text{ and } O_n \text{ such that } \exists e_{i,n} \in \mathcal{E}. \tag{3.4}$$

If O_n is not scheduled immediately after the completion of O_i then a storage module is required. The number of such storage modules during a time step j is important in defining the placement constraints for the model, since the storage modules also occupy chip area. Using a binary variable $m_{i,j}$ defined as:

$$m_{i,j} = \begin{cases} 1, & \text{if a storage unit is needed for } O_i \text{ at time step } j \\ 0, & \text{otherwise} \end{cases}$$

we can capture if a storage is required for operation O_i at time step j. The binary variable associated with the edge between O_i and O_n is expressed as:

$$\sum_{h=1}^{j-C_i^{M_k}} \sum_k \sum_l z_{i,h,k,l} - \sum_{h=1}^{j} \sum_k \sum_l z_{n,h,k,l} = m_{i,j}, \quad \forall j \in \mathcal{T}, \forall O_i, O_n \in \mathcal{V} \text{ such that } \exists e_{i,n} \in \mathcal{E} \tag{3.5}$$

Variable $m_{i,j}$ will have the value 1 at that time step j when O_i has finished executing (first sum of the equation equals 1), but O_j has not started yet (second term of the equation is 0). Let us define s_j as the number of storages that need to be placed at time step j.

Based on the variable $m_{i,j}$ we can express the synthesis of storage operations at time step j as follows:

$$z_{i,j,k,l} = m_{i,j}, \quad \forall O_i \in \mathcal{S}t, \forall j \in \mathcal{T}. \tag{3.6}$$

Therefore, an additional operation will be activated at time step j only if the operation in $\mathcal{V}$ to which it is associated requires a storage unit.

However, defining a storage for each operation in the graph leads to an explosion of the ILP exploration space. In order to reduce the time taken by the ILP in obtaining the optimal solution, we have used a simplified synthesis of the storage operations in our experimental results. We have reduced the number of additional operations defined in $\mathcal{S}t$ to an upper boundary determined by the number of operations in the graph and the biochip specifications.

At each time step we can determine the number of storages that need to be scheduled and placed on the array, s_j as:

$$s_j = \sum_i m_{i,j}, \quad \forall O_i \in \mathcal{V}. \tag{3.7}$$

Knowing the number of storages, we know how many additional operations need to be executed at time step t:

$$z_{i,j,k,l} = 1, \quad \forall O_i \in \mathcal{S}t \text{ such that } i \ \leq s_j, \forall j \in \mathcal{T}. \tag{3.8}$$

thus only the first s_j operations in the set $\mathcal{S}t$ will be scheduled for execution.

Each store operation will be bound to a 1 $\times$ 1 module (3 $\times$ 3 with segregation area).

3.2.2 Resource constraints

Considering the fact that two non-reconfigurable operations of the same type can be bound to the same resource, a constraint must be expressed to prevent the overlapping of these operations during their execution. An operation $O_i \in \mathcal{V}^{nonReconf}$ is executing at time step j if:

$$\sum_{h=j-C_i^{M_k}+1}^{j} \sum_{k} \sum_{l} z_{i,h,k,l} = 1, \quad \forall O_i \in \mathcal{V}^{nonReconf}.$$

Thus, at any time step $j \in \mathcal{T}$ there most be at most one non-reconfigurable operation O_i executing on module M_k:

$$\sum_{i} \sum_{h=j-C_i^{M_k}+1}^{j} \sum_{l} z_{i,h,k,l} \leq 1, \quad \forall M_k \in \mathcal{L}, j \in \mathcal{T}. \tag{3.9}$$

3.2.3 Placement Constraints

We consider two different constraints for the placement problem. The first constraint (represented by equation 3.10) can be used for simplified placement when separating the architectural-level and placement steps during the synthesis process, as explained in Section 3.1.1. The constraint can be used as guidance while performing the allocation, binding and scheduling steps during the synthesis process. At each time step j, the sum of the modules scheduled to be placed should not exceed the total area size of the array, m × n. As input ports are placed outside the microfluidic array we consider their dimensions (width and length) as zero during the placement step.

$$\sum_{i} \sum_{h=j-C_i^{M_k}+1}^{j} \sum_{k} \sum_{l} z_{i,h,k,l} \times L_k \times W_k \leq m \times n, \forall j \in \mathcal{T} \tag{3.10}$$

where $O_i \in \mathcal{V}^{+}$ and L_k and W_k are the length and width of module M_k, respectively, measured in number of cells.

However, this constraint does not ensure that all the modules bound to operations scheduled at a time step t can be placed on the microfluidic array without

overlapping. Thus, equation 3.10 can only be used as an estimate of the area occupied by modules in the case when architectural-level synthesis is performed before the placement step.

For unified architectural-level synthesis and placement equation 3.11 must be used. In order to ensure that all the modules bound to operations scheduled at time step t can be placed on the biochip without overlapping we place the constraint that a cell c_l on the array can be occupied by at most one module during time step t_j.

Let us consider a cell c_r (with coordinates x_r and y_r) which is the bottom-left corner of module M_k. If cell c_l is within the rectangle formed by M_k, i.e., $x_r \leq x_l \leq x_r + L_k - 1$ and $y_r \leq y_l \leq y_r + W_k - 1$, then we have to impose the restriction that no other module is active during this time interval:

$$\sum_{i} \sum_{h=j-C_i^{M_k}+1}^{j} \sum_{k} \sum_{r} z_{i,h,k,r} \leq 1, \forall j \in \mathcal{T}, \forall c_l \tag{3.11}$$

where $O_i \in \mathcal{V}^+$.

The complexity of the ILP formulation is $O(|\mathcal{V}^+|\,|\mathcal{L}|\,|\mathcal{T}|\,mn)$ in the number of variables and $O(mn\,|\mathcal{T}| + |\mathcal{T}|\,|\mathcal{E}| + 3\,|\mathcal{V}^+| + |\mathcal{E}| + 2\,|\mathcal{T}|)$ in the number of constraints, where $\mathcal{V}^+$ is the set of all operations to be scheduled, $\mathcal{L}$ represents the module library, $\mathcal{T}$ is the set of time steps, $\mathcal{E}$ is the set of dependencies between the operations in the graph and m and n represent the two dimensions of the microfluidic array.

3.3 Experimental Evaluation

We have used the ILP formulation presented in Section 3.2 to evaluate the advantages of performing unified architectural-level synthesis and placement. For this purpose, we used two real-life examples: 1) *In-vitro* diagnostics on human physiological fluids (IVD) (see Section 2.4.2)[2] 2) The mixing stage of a polymerase chain reaction application (PCR/M), see Section 2.4.1.

We have solved the ILP model with GAMS 21.5 using the CPLEX 9.130 solver. All the experiments were ran on Sun Fire v440 computers with 4 UltraSPARC IIIi CPUs at 1,062 MHz and 8 GB of RAM. The results are presented in Table 3.3.

[2]The input and detection operations were ignored.

Operation	Area (cells)	Time (s)
Mixing	2×5	2
Mixing	2×4	3
Mixing	1×3	5
Mixing	3×3	7
Mixing	2×2	10
Dilution	2×5	4
Dilution	2×4	5
Dilution	1×3	7
Dilution	3×3	10
Dilution	2×2	12
Detection	1×1	30
Dispensing	-	7
Storage	1×1	-

Table 3.2: Table for experimental evaluation

For each application, we have considered the library from Table 3.2 and three, progressively smaller, area constraints (second column of Table 3.3). We have performed the allocation, binding, scheduling and placement such that the application completion time is minimized. We used two approaches to derive the implementations:

1. The *straight-forward* approach (SF) does architectural synthesis (allocation, binding and scheduling) separately from placement. First, an implementation Ψ^0 is derived using our ILP model, limited by the total chip area, but without the placement constraints. Next, we attempt the placement of Ψ^0 on the available area, modifying the scheduling, if required to fit the modules, thus obtaining the final implementation Ψ. For both of these steps we have obtained the optimal solutions. The schedule length of Ψ is presented in column 3.

2. The *Optimal Synthesis* approach (OS) performs unified architectural-level synthesis and placement. The schedule length for OS is presented in column 4, with the execution time required by the CPLEX solver to produce the optimal solution shown in column 5.

As we can see from Table 3.3, considering placement at the same time with architectural synthesis (OS) can lead to significant improvements over SF, which does not take into account placement. On average, we have obtained an 11.8% improvement on the bio-applications completion times, with up to 18.75% improvement for the IVD application on a 7×9 array.

Application	Area	SF	OS	Execution Time
IVD	8 × 9	14 s	13 s	90 min
	7 × 9	16 s	13 s	258 min
	7 × 8	14 s	14 s	78 min
PCR/M	8 × 9	11 s	9 s	84 min
	7 × 9	11 s	10 s	47 min
	7 × 7	17 s	14 s	78 min

Table 3.3: Comparison of SF and OS approaches

The synthesis problem presented in this chapter is NP-complete. Scheduling in even simpler contexts is NP-complete [64]. In addition, the placement is equivalent to a sequence of 2D packing problems, known to be NP-complete [16]. Using ILP, we have not been able to synthesize larger examples than IVD and PCR/M. Even for these two cases, the runtime of the solver has been quite large, see the last column in Table 3.3. The reason is that the ILP formulation searches for the optimal solution, and considering the complexity of the synthesis problem, this is not feasible for larger examples.

This has motivated us to use instead a metaheuristic such as Tabu Search as the basis for our synthesis strategies presented in this thesis.

Chapter 4

Related Work

In the recent years there has been a growing interest in the development of CAD tools for digital microfluidic biochips. As the complexity of such devices increases, with more and more operations being executed concurrently on the chip, top-down synthesis approaches are required. Although design automation techniques are essential for the further development of biochips, CAD tools for such devices are still in their infancy. This chapter presents a brief overview of the related research on the digital microfluidic biochips, with an emphasis on architectural and physical-level synthesis of direct-addressable chips.

4.1 Architectural-Level Synthesis and Placement

Researchers have initially addressed separately architectural- and physical-level synthesis of DMBs. In one of the first papers on this topic Ding et al. [10] have proposed an architectural design and optimization method for performing biochemical assays on a digital biochip. The method is based on ILP and it aims at improving scheduling by extracting the parallelism from a biochemical application, thus performing several operations in parallel.

In [53] Su et al. have proposed an ILP and two heuristic techniques (a modified List Scheduling algorithm and a Genetic Algorithm) for the architectural-level

synthesis of biochips. The proposed methods are considering the problem of scheduling under resource constraints. The number of reconfigurable devices that can be placed on the array concurrently is roughly estimated by a simplified placement, as explained in Chapter 3.

The results in [53] have been improved by Ricketts et al. [47], by using a hybrid Genetic Algorithm for scheduling operations under resource constraints. In their work, the operations that are bound to the same device are considered to form a group, competing for the access to the same resource. Conflicts between operations are handled by reserving a resource to the highest priority operation ready to execute.

A Simulated Annealing-based method which can determine the locations of the devices on the microfluidic array has been proposed by Su et al. in [55]. The algorithm follows architectural-level synthesis, thus the binding of operations to modules and the schedule of operations are given as an input. As detailed placement information is not considered during the scheduling step, there is no guarantee that all modules can be placed at time t without overlapping. Therefore, the algorithm seeks to optimize the design metrics (area of the microfluidic array and fault-tolerance) while minimizing the overlapping of modules.

Although it reduces the complexity of the synthesis problem, the separation of architectural and physical-level synthesis has disadvantages, leading in many cases to a longer completion time of the applications on the biochips (see Chapter 3). Therefore, the next step taken by researchers was considering a unified approach for the architectural-level synthesis and placement for digital microfluidic biochips.

The first unified high-level synthesis and module placement methodology has been proposed by Su et al. in [54], by using a combination of Simulated Annealing and Genetic Algorithms. In their work, the characteristics of a candidate solution is encoded in a chromosome, where each operation is randomly bound to a device in the module library. The schedule of operations is determined using a List Scheduling algorithm while the placement of modules on the array is performed in a greedy fashion. The focus of the developed methodology has been on deriving an implementation that can tolerate faulty electrodes.

The results obtained in [54] have been improved by using another unified approach, proposed by Yuh et al. [74]. The algorithm is based on the T-Tree, a data structure in which each node represents an operation and has at most three children. The order of the nodes in the tree is based on a geometric relationship between the operations, e.g., if a node n_i is the left child of node n_j then the module M_i will be placed adjacent to module M_j. In order to improve the completion time of the application Simulated Annealing is used to perturb the initial

T-Tree and to explore the search space for better solutions. The floorplanning algorithm is also extended to take into account the reconfigurability of biochips in case of defective electrodes. In Chapter 5 we have proposed a Tabu Search-based algorithm for the unified synthesis problem. Our method can produce improvements of up to 22% compared to the T-Tree approach from [74].

Xu et al. [70] have extended the work done by [54], by incorporating routing-awareness during the architectural-level synthesis and placement of modules. Droplet routability is defined by the ease with which routing can be performed once the placement of the modules on the array has been determined. Although it does not offer detailed routing information, the aim of this approach is to construct synthesis solutions that lead to simpler droplet routes. For example, if a droplet needs to be routed between two modules, it is desirable that the modules are placed such that the route is minimized. In their work, the route is estimated as the shortest distance between modules, assuming no obstacles between them. Overall, routability is evaluated by estimating the average length of all the droplet routes for a given chip.

4.2 Routing

Another important step during the synthesis problem is determining the droplet routes between modules and between I/O ports and modules. Due to the complexity of the problem and long operation execution times, routing has been addressed so far as a post-synthesis step, following the placement of modules on the array. Several techniques have been proposed for finding the routes on which droplets are transported.

In [2] Bohringer et al. have presented a methodology for routing droplets between two given points, in the shortest number of steps. The proposed algorithm is based on a graph data structure, where a node represents the state of the microfluidic array at time t. Therefore, routing is transformed into a standard graph search problem, where the start and goal states are given. A prioritized A* search algorithm is used for determining the routes, at each step the optimal motion plan being performed for the droplet with the highest priority.

In [57], a two-stage routing algorithm has been proposed by Su et al. Following the module placement step, routing is decomposed into a series of sub-problems, in which droplet paths must be determined. During the first stage a set of alternative routes are generated for each net using a modified Lee algorithm, while in the second stage a single route is randomly selected. Routes are checked for fluidic constraints in order to ensure that the droplets will not accidentally

merge during the routing step. The output of the algorithm consists in a set of routes with minimum lengths, which ensure the fluidic constraints.

Griffith et al. have proposed a routing method based on the Open Shortest Path First network protocol [20]. In their approach the microfluidic array is partitioned into virtual components (e.g., work area = mixers, source = input points for reservoirs), each performing a specific set of operations. In order to determine droplet routes between components, the proposed algorithm uses routing tables, computed using Dijkstra's shortest path algorithm.

A network-flow based method for the routing problem has been proposed by Yuh et al. in [73]. The algorithm consists in three steps: 1) determining the criticality of each net; 2) finding a rough routing path for each droplet; and 3) routing the droplets in the decreasing order of their criticality. During the second step the microfluidic array is divided into a set of global electrodes (obtained by grouping together 3×3 basic electrodes), on which the flow network is constructed.

The results obtained in [73] have been improved by Cho et al. in [7] by performing bypassability analysis while routing. Each droplet is assigned a priority, which indicates how likely it is that its routing will block the movement of other droplets on the array. The droplets with higher bypassability are routed first, decreasing the chances of deadlocks. If, however, a deadlock is created, the algorithm uses concession zones in which droplets can be moved in order to eliminate deadlock.

In [66] a routing algorithm for cross-referencing biochips has been proposed by Xiao et al. The routing paths are determined by using a weighted maze framework, which determines for each droplet a valid shortest path. If no valid route can be found, backtracking and re-routing are considered. The voltage assignment for controlling the electrodes is performed based on the routing result.

All of the synthesis approaches developed so far have considered that modules are fixed during their execution and have a rectangular shape. In Chapters 6–8 we have relaxed this assumption, and have presented synthesis approaches which are based on a routing model, where the operation execution is seen as a "route", even for the case where it is constrained to fixed rectangular shapes (Chapter 7).

We have shown that significant improvements can be gained by using such a routing-based model of operation execution. All the synthesis methods presented in Chapters 6–8 are able to improve on the results in Chapter 5.

4.3 Cross-Contamination Avoidance

As explained in Chapter 2, the contamination of the microfluidic array can have serious consequences on the output of a biochemical application. Several algorithms for cross-contamination avoidance have been recently proposed by researchers.

The first routing method which considers cross-contamination avoidance has been proposed in [75] by Zhao et al. The routing problem is divided into a set of sub-problems, based on the results from the placement step. The algorithm tries to minimize contamination in a sub-problem by finding disjoint droplet routes. As contamination can also occur between successive sub-problems, wash droplets are introduced after each sub-problem to remove the residue left on electrodes. An optimization method is presented to minimize the number of used wash droplets.

Huang et al. have proposed in [22] a contamination-aware droplet routing algorithm. The method minimizes the contaminated spots by constructing preferred routing tracks on which droplets are routed. A minimum cost circulation algorithm is used for simultaneously cleaning the contaminated electrodes inside a sub-problem and between successive sub-problems.

Another method for cross-contamination avoidance routing has been presented by Zhao et al. in [76]. The algorithm improves on the results obtained in [75] by integrating the required washing operations in each sub-problem. The routing time is reduced by synchronizing the arrival time of wash droplets and other droplets on the array to the contaminated spots.

A design flow that considers the cross-contamination problem on pin-constrained biochips has been proposed by Lin et al. in [28]. The reduction in droplet movement flexibility compared to direct-addressing biochips adds to the complexity of contamination avoidance on such chips. Compared to previous works, the authors consider crossing minimization earlier in the synthesis process, during the placement step. The insertion of wash droplets for cleaning contaminated electrodes is done using only one extra control pin.

Several other techniques for pin-constrained microfluidic biochips [23],[67], [68], [72], [29] and testing [59], [60], [61], [58], [25], [69], [71], have been proposed recently. However, these topics are not part of the research work presented in this thesis. For further details the reader is directed to the cited publications.

In Chapter 8 we show how contamination avoidance can be taken into account during routing-based synthesis.

Chapter 5

Module-Based Synthesis

Although capable of producing an optimal solution, the ILP-based method presented in Chapter 3 becomes unfeasible for complex synthesis problems, where the increase in the search space leads to a prohibitively large execution time of the program. Therefore, in this chapter we present a methodology based on the Tabu Search (TS) metaheuristic, capable of solving complex synthesis problems. Similar to Chapter 3, we consider that reconfigurable operations are performed inside fixed rectangular modules.

Our synthesis strategy, presented in Figure 5.1, takes as input the application graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the given biochip cell array $\mathcal{C}$ and the module library $\mathcal{L}$ and produces that implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P} >$ consisting of, respectively, the allocation $\mathcal{A}$, binding $\mathcal{B}$, scheduling $\mathcal{S}$ and placement $\mathcal{P}$, which minimizes the schedule length $\delta_{\mathcal{G}}$ on the given biochip $\mathcal{C}$. A Tabu Search metaheuristic [18]

DMBSynthesis($\mathcal{G}$, $\mathcal{C}$, $\mathcal{L}$)
1 $< \mathcal{A}^{\circ}, \mathcal{B}^{\circ} >$ = InitialSolution($\mathcal{G}$, $\mathcal{L}$)
2 Π° = CriticalPath($\mathcal{G}$, $\mathcal{A}^{\circ}$, $\mathcal{B}^{\circ}$)
3 $< \mathcal{A}, \mathcal{B}, \Pi >$ = TabuSearch($\mathcal{G}$, $\mathcal{C}$, $\mathcal{L}$, $\mathcal{A}^{\circ}$, $\mathcal{B}^{\circ}$, Π°)
4 **return** $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P} >$

Figure 5.1: Synthesis algorithm for DMBs

is used for deciding the allocation $\mathcal{A}$ and binding $\mathcal{B}$ of operations (line 3 in Figure 5.1). For a given allocation and binding decided by TS, we use a List Scheduling (LS) heuristic [36] to decide the schedule $\mathcal{S}$ of the operations. LS uses the priorities Π (assigned to each operation) to decide which operation to schedule at a given time step, out of several "ready" operations competing for the same resource. Our TS also decides the priorities Π for the operations. TS starts from an initial solution, where we consider that each operation $O_i \in \mathcal{V}$ is bound to a randomly chosen module $\mathcal{B}(O_i) \in \mathcal{L}$ (line 1 in Figure 5.1). The initial execution priorities, Π°, are given according to the bottom-level values of the nodes in the graph (line 2) [49]. According to these, the priority of an operation is defined as the length of the longest path from the operation to the sink node of the graph.

5.1 List Scheduling

Inside TS, we use the *ScheduleAndPlace* function in Figure 5.2 to determine the schedule $\mathcal{S}$ and placement $\mathcal{P}$ for an implementation Ψ. Our scheduling is based on a List Scheduling heuristic, takes as input the application graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the cell array $\mathcal{C}$, the allocation $\mathcal{A}$, binding $\mathcal{B}$ and priorities Π and returns the scheduling $\mathcal{S}$ and placement $\mathcal{P}$. The List Scheduling heuristic is based on a sorted priority list, L_{ready}, containing the operations $O_i \in \mathcal{V}$ which are ready to be scheduled. The start and finish times of all the operations are initialized to 0 in the beginning of the algorithm (lines 2 and 3 in Figure 5.2). A list $L_{execute}$ which contains the operations that are executing at the current time step is created in the beginning of the algorithm (line 4). Initially, L_{ready} will contain those operations in the graph that do not have any predecessors (line 5 in Figure 5.2). We do not consider input operations as part of the ready list. As they do not have any precedence constraints, input operations can be executed at any time step. However, it is important that inputs and their successors are performed sequentially, in order to avoid storing the dispensed droplets. Let us consider time $t_{current}$ during the execution of the application. For all the operations that finish executing at $t_{current}$ we check if their successors are ready to be scheduled (line 14 in Figure 5.2). An operation is considered to be ready if all its predecessors (except input operations) have finished executing. Next, we try and schedule the ready operations, starting with the operation O_j having the highest priority (line 17 in Figure 5.2). Before O_j can be scheduled, its input constraints must be checked. If O_j has as predecessor an input operation O_k, we try to schedule O_k such that $t_k^{finish} = t_j^{start} = t_{current}$. However, as reservoirs/dispensing ports are non-reconfigurable devices, their number is constrained during design specifications. That is, operation O_j can be scheduled at time step t only if at time $t - C_k^{reservoir}$ there is an available reservoir/dispensing

ScheduleAndPlace($\mathcal{G}$, $\mathcal{C}$, $\mathcal{A}$, $\mathcal{B}$, Π)

```
1  t_current = 0
2  t_i^start = 0, ∀O_i ∈ G
3  t_i^finish = 0, ∀O_i ∈ G
4  L_execute = ∅
5  L_ready = ConstructReadyList(G, Π)
6  // schedule and place operations
7  while ∃ O_i ∈ G ∧ t_i^finish = 0 do
8     // for finishing operations
9     for all O_j ∈ L_execute such that t_j^finish = t_current do
10       // update placement
11       UpdatePlacement(C, P, B(O_j))
12       RemoveFromExecuteList(O_j, L_execute)
13       // add ready successors to L_ready
14       AddReadySuccessorToList(O_j, L_ready)
15    end for
16    // schedule ready operations
17    for all O_j ∈ L_ready do
18       placed = Placement(C, P, B(O_j))
19       if placed then
20          // set the start and finish times
21          t_j^start = t_current
22          t_j^finish = t_j^start + C_j^B(O_j)
23          RemoveFromReadyList(O_j, L_ready)
24          AddOperationToExecuteList(O_j, L_execute)
25       end if
26    end for
27    t_current = t_current + 1
28 end while
29 return < S, P >
```

Figure 5.2: List scheduling algorithm for DMBs

port on which O_k can be executed. Otherwise, O_j will be delayed and the next highest priority operation is considered for execution. If all the constraints related to O_j are satisfied, its corresponding module, $\mathcal{B}(O_j)$, is placed on the microfluidic array (line 18 in Figure 5.2) and the start and finish times of the operation are updated (lines 21–22). If there exists a placed storage module associated with the operation O_j, the storage is removed and the placement is updated.

The combined scheduling and placement is implemented by the *ScheduleAndPlace* function (Figure 5.2), which calls the *Placement* function from Figure 5.3. Once an operation is scheduled it is removed from L_{ready} and added to $L_{execute}$. Before the end of the iteration, the storage constraints are considered. For all the operations that finished at $t_{current}$ the placement of the microfluidic array must be updated, by removing the modules to which they are bound (line 11 in Figure 5.2). Also, if their successors have not yet been scheduled for execution, a storage unit is placed on the microfluidic array.

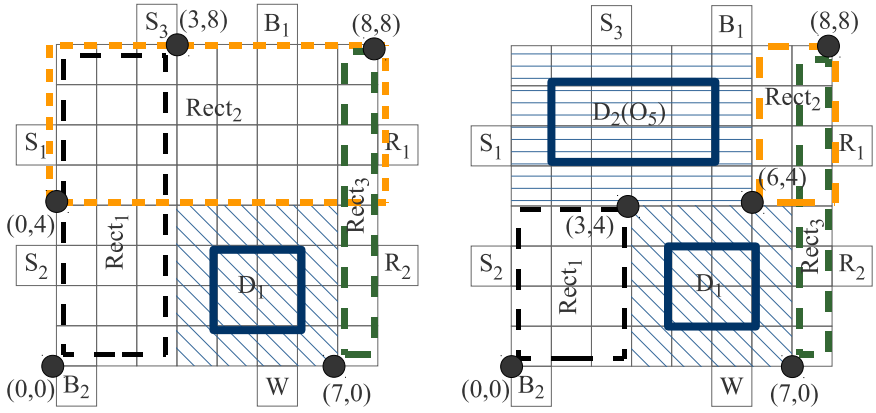
5.2 Placement Algorithm

We have used the online placement algorithm from Bazargan et al. [1] for DR-FPGAs to handle the placement of modules in DMBs. Although the algorithm was proposed for online placement, we can use it offline, since we know beforehand all the operations that have to be executed. The algorithm from [1] has two parts: i) a free space manager which divides the free space on the biochip into a list of overlapping rectangles, L_{rect}; and ii) a search engine which selects an empty rectangle from L_{rect} that best accommodates the module M_i to be placed, according to a given criteria, such as "best fit". Each rectangle

Placement($\mathcal{C}$, $\mathcal{P}$, M_i)

```
1 // construct list of empty rectangles
2 L_rect = ConstructRectList(C)
3 // search for R_i ∈ L_rect that best fits M_i
4 R_i = SelectRectangle(L_rect, M_i)
5 if ∃ R_i then
6    placed = UpdatePlacement(P, R_i, M_i)
7    UpdateFreeSpace(L_rect)
8 end if
9 return placed
```

Figure 5.3: Placement algorithm for DMBs



(a) Placing $Diluter_2$ (b) Placement at t

Figure 5.4: Placement example

is represented by the coordinates of its left bottom and right upper corners, (x_l, y_l, x_r, y_r). Our proposed algorithm, *Placement*, is presented in Figure 5.3. The placement algorithm takes as input the $m \times n$ matrix $\mathcal{C}$ of cells, the current placement of modules $\mathcal{P}$ and the module M_i to be placed, updates the array and returns a Boolean value stating if the accommodation of module M_i on the array was successful. If no rectangle is found, LS will have to delay the operation corresponding to M_i.

Let us illustrate the placement algorithm by using the same example as in Section 3.1.3. Consider the current time step t. The ready list consists of all the operations in the graph that are ready to be scheduled, hence $L_{ready} = \{O_5, O_6, O_{12}, O_{13}\}$. Let us assume the same binding for the reconfigurable operations as the one shown in Figure 3.3, thus O_5 is bound to a 2×4 diluter, O_6 to a 2×2 mixer, O_7 to a 2×4 mixer and O_{12} and O_{13} to 2×2 diluters. Then the priorities of the operations ready to be scheduled are computed as follows: $\pi_{O_5} = \pi_{O_6} = \max(C_{O_5}^{Diluter_2}, C_{O_6}^{Mixer_1}) + C_{O_7}^{Mixer_2} = 4 + 3 = 7$, $\pi_{O_{12}} = C_{O_{12}}^{Diluter_3} = 5$, $\pi_{O_{13}} = C_{O_{13}}^{Diluter_4} = 5$.

Accordingly, the LS algorithm will select O_5 and will try and schedule its predecessor input operations, O_1 and O_2 such that they finish executing at the current time step. As there are available reservoirs on which O_1 and O_2 can be executed, LS will schedule O_5 and will call *Placement* to place $Diluter_2$ on the biochip array. The module $Diluter_1$, which is currently executing at time t, divides the free space into three overlapping rectangles $L_{rect} = \{Rect_1 = (0, 0, 3, 8), Rect_2 = (0, 4, 8, 8), Rect_3 = (7, 0, 8, 8)\}$, see Figure 5.4a (line 2 in Figure 5.3). As rectangle $Rect_2 = (0, 4, 8, 8)$ is the only one sufficiently large to accommo-

date the 2 × 4 module (line 4), $Diluter_2$ will be placed at its bottom corner (line 6 in Figure 5.3). Consequently, in line 7, the free space will be updated to $L_{rect} = \{Rect_1 = (0, 0, 3, 4), Rect_2 = (6, 4, 8, 8), Rect_3 = (7, 0, 8, 8)\}$ as depicted in Figure 5.4b.

After the scheduling and placement of O_5, the next operation to be considered for scheduling at time t is O_6. Because of space fragmentation, no free rectangle can accommodate the 2 × 2 mixer currently assigned to O_6 and the operation will have to be delayed until $t + 4$ (see Figure 3.3a), when the module denoted by D_1 is removed from the array.

Placement of Non-Reconfigurable Devices

The placement of a non-reconfigurable device (e.g., an optical detector) on the microfluidic array is similar to that of a reconfigurable module. However, once decided, the location of the device remains fixed throughout the execution of the application. Therefore, our algorithm maintains a list of locations at which non-reconfigurable operations of each type (e.g., detection operations) can be performed, $L_{nonReconf}$. These locations are established during the execution of the placement algorithm. The size of the list is constrained by the maximum number of devices of the given type that can be integrated on the chip, given as an input during design specifications. Let us consider that at time t a non-reconfigurable detection operation is ready to be scheduled. We try and place the 3 × 3 detector at one of the locations in L_{detect}. If no locations have been established previously or if they are all occupied but we can still integrate detectors on the array, we use the algorithm in Figure 5.3 to find a new detector location. If a free rectangle that can accommodate the 3 × 3 module is found, the operation is scheduled at time step t and the point corresponding to the left bottom corner of the rectangle is added to L_{detect}. Otherwise the detection operation can not be scheduled at time t. Just as in the case of reconfigurable modules, non-reconfigurable devices can not overlap with other modules placed on the chip.

5.3 Tabu Search

Tabu Search [18] is a metaheuristic based on a neighborhood search technique which uses design transformations (moves) applied to the current solution, $\Psi^{current}$, to generate a set of neighboring solutions, N, that can be further explored by the algorithm. Our TS implementation performs two types of transformations: i) rebinding moves and ii) priority swapping moves. A rebinding move consists in the rebinding of a randomly chosen operation, O_i, currently

executing on module M_i, to another module M_j. Such a move will take care of the allocation, e.g., removing M_i and allocating M_j. A priority swapping move consists in swapping the priorities of two randomly chosen operations in the graph.

In order to efficiently perform the search, TS uses memory structures, maintaining a history of the recent visited solutions (a "tabu" list). By labeling the entries in the list as tabu (i.e., forbidden), the algorithm limits the possibility of duplicating a previous neighborhood upon revisiting a solution.

However, in order not to prohibit attractive moves, an "aspiration criteria" may be used, allowing tabu moves that result in solutions better than the currently best known one. Moreover, in order to avoid getting stuck in a local optima, TS uses "diversification". This involves incorporating new elements that were not previously included in the solution, in order to diversify the search space and force the algorithm to look in previously unexplored areas.

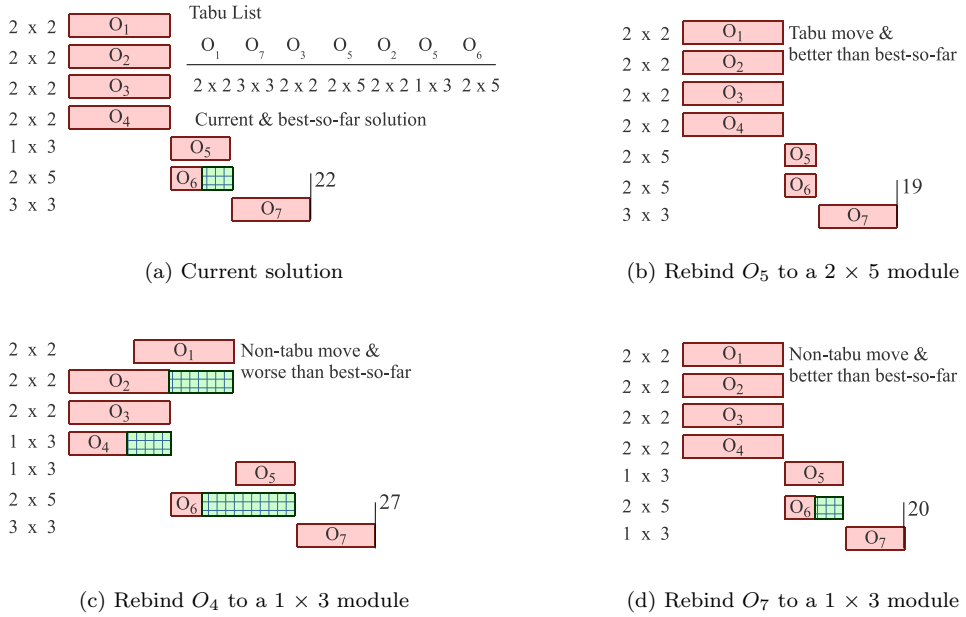
Our algorithm uses two tabu lists, one for each type of move. These are constructed as attribute-based memory structures, containing the relevant modified attributes. Hence, if an operation O_i is rebound to a module M_j as result of a rebinding move, the change of the solution will be recorded in the corresponding tabu list as a pair of the form (O_i, M_j) and if the priorities of two operations O_i and O_j are swapped as part of the diversification process, the move will be recorded as an entry of the form (O_i, O_j). Based on experiments, we have decided to use priority swapping as a diversification move, only when the best known solution does not improve for a defined number of iterations, num_{div}, determined experimentally.

The TS algorithm presented in Figure 5.5 takes as input the application graph $\mathcal{G}$, the biochip array $\mathcal{C}$, the module library $\mathcal{L}$ and the initial allocation $\mathcal{A}^\circ$, binding $\mathcal{B}^\circ$ and priorities Π° and returns the best implementation Ψ^{best} found over a number of iterations. TS starts from an initial solution Ψ° where each operation is bound to a randomly chosen module and has a priority given according to the bottom-level value of the corresponding node in the graph. The schedule $\mathcal{S}^\circ$ and placement $\mathcal{P}^\circ$ for the initial solution are obtained by using the *ScheduleAndPlace* function (line 1). Knowing the schedule $\mathcal{S}^\circ$, the initial schedule length, $\delta^\circ_{\mathcal{G}}$ can be determined (line 3). Two tabu lists, $tabuList_{dev}$ and $tabuList_{prio}$ are used for recording the rebinding moves, respectively the priority swapping moves. Each list has a given size, $tabuSize_{dev}$ and $tabuSize_{prio}$ correspondingly, specifying the maximum number of moves that can be recorded. Initially, the lists are empty (lines 4–5). A variable num_{iter} is used to keep track of the number of iterations passed without the improvement of the best solution, $\Psi^{best}_{\mathcal{G}}$ (line 6). The algorithm is based on a number of iterations (lines 7–29 in Figure 5.5) during which the aim is improving the overall best solution $\Psi^{best}_{\mathcal{G}}$. In each iteration, a set

TabuSearch($\mathcal{G}$, $\mathcal{C}$, $\mathcal{L}$, $\mathcal{A}^\circ$, $\mathcal{B}^\circ$, Π°)

```
1  < S°, P° > = ScheduleAndPlace(G, C, A°, B°, Π°)
2  Ψ^best = Ψ^current = Ψ° = < A°, B°, S°, P° >
3  δ_G^best = δ_G^current = δ_G° = GetCompletionTime(S°)
4  tabuList_dev = ∅
5  tabuList_prio = ∅
6  num_iter = 0
7  while timeLimit not reached do
8     N = GenerateNeighborhood(Ψ^current, L)
9     Ñ = SelectAllowedMoves(N)
10    (O_i, B(O_i)) = SelectBestMove(Ñ)
11    PerformBestMove(Ψ^current, O_i, B(O_i))
12    RecordRebindMove(O_i, B(O_i), tabuList_dev)
13    δ_G^current = GetCompletionTime(S^current)
14    if δ_G^current < δ_G^best then
15       Ψ^best = Ψ^current; δ_G^best = δ_G^current
16    else
17       num_iter = num_iter + 1
18       if num_iter = num_div then
19          (O_i, O_j) = SelectSwapMove(G, Π^current, tabuList_prio)
20          PerformSwapMove(Ψ^current, O_i, O_j)
21          RecordSwapMove(O_i, O_j, tabuList_prio)
22          δ_G^current = GetCompletionTime(S^current)
23          if δ_G^current < δ_G^best then
24             Ψ^best = Ψ^current; δ_G^best = δ_G^current
25          end if
26          num_iter = 0
27       end if
28    end if
29 end while
30 return Ψ^best
```

Figure 5.5: Tabu Search algorithm for DMBs



(a) Current solution

(b) Rebind O_5 to a 2×5 module

(c) Rebind O_4 to a 1×3 module

(d) Rebind O_7 to a 1×3 module

Figure 5.6: Tabu Search neighborhood

of possible candidates N is obtained by applying moves to the current solution, $\Psi^{current}$ (line 8). However, N might contain solutions that are disallowed by TS. According to the aspiration criteria, a tabu move $(O_i, M_j) \in tabuList_{dev}$ is only allowed if it leads to a solution better than the currently best known one. Therefore, all the tabu moves resulting in solutions with schedule lengths $\delta_{\mathcal{G}}^{current}$ worse than the currently best one are removed from N and thus, the set $\tilde{N}$ of allowed moves is created (line 9). The *ScheduleAndPlace* function is used for determining the move $(O_i, M_j) \in \tilde{N}$ leading to the solution with the shortest schedule length $\delta_{\mathcal{G}}^{current}$ among all the moves in $\tilde{N}$. The move is selected and marked as tabu (lines 10–12). If the obtained solution has a better schedule length that the currently known one, the best-so-far solution is updated (lines 14–15). When the best known solution does not improve for a given number of iterations num_{div} a diversification move is considered (line 18), forcing the search into unexplored areas of the search space. The move consists in swapping the priorities of two randomly selected operations O_i and O_j, with $(O_i, O_j) \notin TabuList_{prio}$. If the move results in a new best known solution, Ψ^{best} is updated to $\Psi^{current}$ (line 23). Finally, the variable num_{iter} is reset to 0 (line 26).

Let us use the mixing stage of the polymerase chain reaction (see Section 2.4.1 for details), and the module library in Table 3.2 to illustrate how TS works. Consider the current solution as being the one represented by the schedule in Figure 5.6a. The current tabu list $tabuList_{dev}$, presented to the right, contains the recently performed transformations. As all operations are mixing operations, we will denote a module by its area, e.g. O_1 is bound to a mixing module of 2×2 cells. Starting from this solution, TS uses rebinding moves to generate the neighbor solutions (line 8 in Figure 5.5). We consider that the number of elements in the neighborhood equals the number of operations in the application graph $\mathcal{G}(V, E)$. Out of the possible neighboring solutions we present three in Figure 5.6b–d. The solution in Figure 5.6b is tabu and the one in Figure 5.6c is worse than the current solution (which is the best so far). In the solution in Figure 5.6d O_7 is rebound to a new, 1×3 mixer, which results in a non-tabu solution better than the current one. However, TS will select the move in Figure 5.6b, that would change the 1×3 mixer in Figure 5.6a for O_5 to a 2×5 mixer module. Although the move $(O_5, 2 \times 5)$ is marked as being tabu, it leads to a better result than the currently best known one and thus, according to the aspiration criteria, it is accepted. The new solution is evaluated by using the unified scheduling and placement algorithm presented before which determines the completion time $\delta_{\mathcal{G}}$ of the application graph $\mathcal{G}$. The algorithm terminates when a given time-limit for performing the search has been reached.

5.4 Experimental Evaluation

In order to evaluate our proposed approach we have implemented the Tabu Search-based algorithm[1] in Java (JDK 1.6), running on SunFire v440 computers with UltraSPARC IIIi CPUs at 1,062 GHz and 8 GB of RAM.

In our first set of experiments we were interested to determine the quality of our TS approach. Using the ILP formulation proposed in Section 3.2, we were able to obtain the optimal solutions for two small real-life applications: the in-vitro diagnostics on human physiological fluids, IVD (see Section 2.4.2) and the mixing stage of a polymerase chain reaction, PCR/M, (see Section 2.4.1). Table 5.1 presents the schedule lengths δ_{TS} and δ_{ILP} obtained for the TS approach, respectively ILP, for the PCR/M and IVD examples, using the module library in Table 3.2. For the comparison we have considered three areas, from $Area_1$ (largest) to $Area_3$ (smallest). As it can be seen, our Tabu Search-based approach is capable of obtaining the optimal solutions for both applications within 1 minute CPU time limit.

[1] Values for TS parameters determined experimentally: no_{div}=7, length of the tabu lists=8.

Application	Area	δ_{ILP}	δ_{TS}	TS exec.time
IVD	8 × 9	13 s	13 s	1 min
	7 × 9	13 s	13 s	1 min
	7 × 8	14 s	14 s	1 min
PCR/M	8 × 9	9 s	9 s	1 min
	7 × 9	10 s	10 s	1 min
	7 × 7	14 s	14 s	1 min

Table 5.1: Comparison of ILP and TS approaches

In our second set of experiments we measured the quality of the TS implementation, that is, how consistently it produces good quality solutions. Hence, we used our TS-based approach for synthesizing a large real-life application implementing a colorimetric protein assay (see Section 2.4.3). The module library used for these experiments is shown in Figure 3.2. Regarding non-reconfigurable constraints, we have considered that at most four optical detectors can be integrated on the chip, together with one reservoir for sample liquid, two for buffer and two for reagent liquid.

Table 5.2 presents the results obtained by synthesizing the protein application on three progressively smaller microfluidic arrays. We present the best solution (in terms of schedule length), the average and the standard deviation obtained after 50 runs of the TS algorithm. Let us first concentrate on the results obtained for the case when we have used a time limit of 60 minutes for the TS. As we can see, the standard deviation is quite small, which indicates that TS consistently finds solutions which are very close to the best solution found over the 50 runs, which will explore differently the solution space, resulting thus in different solutions.

Area	Time limit	Best	Average	Standard dev.
13 × 13	60 min	182	189.88	2.90
	10 min	182	192.00	3.64
	1 min	191	199.20	4.70
12 × 12	60 min	182	190.86	3.20
	10 min	185	197.73	6.50
	1 min	193	212.62	10.97
11 × 12	60 min	184	192.50	3.78
	10 min	194	211.72	14.37
	1 min	226	252.19	15.76

Table 5.2: Results for the colorimetric protein assay

Operation	Area (cells)	Time (s)
Mix	2×4	3
Mix	2×3	6
Mix	2×2	10
Mix	1×4	5
Dilution	2×4	5
Dilution	2×3	8
Dilution	2×2	12
Dilution	1×4	7
Dispensing	–	7
Detection	1×1	30

Table 5.3: Module library for the colorimetric protein assay

Moreover, the quality of the solutions does not degrade significantly if we reduce the time limit from 60 minutes to 10 minutes and 1 minute. This is important, since we envision using TS for architecture exploration, where several biochip architectures have to be quickly evaluated in the early design phases (considering not only different areas, but also different placement of non-reconfigurable resources such as reservoirs or detectors).

In the third set of experiments we have compared our TS-approach with the current state of art for the architectural-level synthesis and placement of DMBs, the T-Tree topological representation [74]. Note that both implementations consider the synthesis of all types of basic microfluidic operations (input, detection, mixing, dilution). We use the same design specifications (resource constraints) as in [74] for the synthesis of two bioassays: the colorimetric protein assay and the in-vitro diagnosis. The module libraries used for the comparison are shown in Tables 5.3 and 5.4. Just like in [74] we assume that for the colorimetric protein assay at most four optical detectors can be integrated on the chip, together with one reservoir for sample liquid, two for buffer and two for reagent liquid. For IVD four samples (S_1 = plasma, S_2 = serum, S_3 = urine and S_4 = saliva) and four reagents (R_1 = glucose, R_2 = lactate, R_3 = pyruvate and R_4 = glutamate) are considered.

Three tests (IVD_1, IVD_2 and IVD_3) are performed, using a different set of samples and reagents, as shown in the first column of Table 5.5. We follow the specifications in [74] and assume that for performing IVD there is one reservoir for each type of samples and reagents and one optical detector for each enzymatic assay. In order to provide a fair comparison, we also consider only one segregation cell between adjacent modules. As [74] considers the area of the biochip as a parameter in the optimization process, we fix the dimensions of

Operation	Area (cells)	Time (s)
Mix (plasma)	2 × 4	3
Mix (plasma)	2 × 3	6
Mix (plasma)	2 × 2	10
Mix (plasma)	1 × 4	5
Mix (serum)	2 × 4	2
Mix (serum)	2 × 3	4
Mix (serum)	2 × 2	6
Mix (serum)	1 × 4	3
Mix (urine)	2 × 4	3
Mix (urine)	2 × 3	5
Mix (urine)	2 × 2	8
Mix (urine)	1 × 4	4
Mix (saliva)	2 × 4	4
Mix (saliva)	2 × 3	8
Mix (saliva)	2 × 2	12
Mix (saliva)	1 × 4	6
Dispensing	–	2
Detection glucose	1 × 1	10
Detection lactate	1 × 1	8
Detection pyruvate	1 × 1	12
Detection glutamate	1 × 1	10

Table 5.4: Module library for in-vitro diagnosis

Application	Area	δ_{TS}	δ_{T-Tree}	Execution time limit
IVD_1	6×9	58	67	9.12
samples: S_1, S_2, S_3, S_4	6×4	92	98	13.22
reagents: R_1, R_2, R_3, R_4	7×4	72	96	10.17
IVD_2	5×4	64	74	7
samples: S_1, S_2, S_3	6×4	52	62	8.28
reagents: R_1, R_2, R_3, R_4	5×4	65	73	10.14
IVD_3	4×4	51	60	3.63
samples: S_1, S_2, S_3	4×4	52	61	4.78
reagents: R_1, R_2, R_3	4×4	60	64	1.72
Protein assay	9×9	187	241	78.03
	10×9	187	211	57.27
	10×10	187	221	68.32
	9×9	185	240	65.21

Table 5.5: Comparison of TS and T-Tree approaches

the microfluidic array and the CPU time to the ones reported by them. Table 5.5 presents the comparison between the TS-based approach and the T-Tree representation for the colorimetric protein and IVD assays. Similar to the experimental setup in [74], the results reported represent the best completion times obtained out of 50 runs for each application. As it can be seen, our TS approach can obtain results up to 22.91% better than the T-Tree representation, for the same design specifications.

Chapter 6

Module-Based Synthesis with Reconfigurable Operation Execution

The algorithms proposed so far for the synthesis problem of digital microfluidic biochips have assumed that reconfigurable operations are performed inside rectangular modules whose location and shape remain fixed throughout the execution of operations. However, as discussed in Section 2.2, reconfigurable operations can be performed anywhere on the array, by simply routing the corresponding droplets on a sequence of electrodes. In this chapter we propose two models for operation execution inside virtual devices, which take into consideration the reconfigurability of microfluidic operations. These models aim at reducing the fragmentation of the free space on the microfluidic array during the placement step of the synthesis process.

6.1 Synthesis with Dynamic Virtual Devices

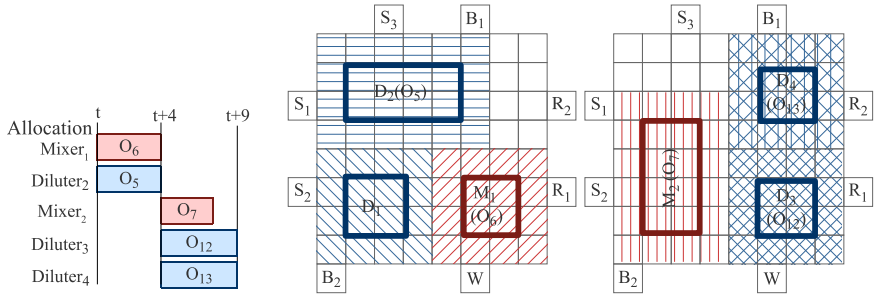
In the first model we consider that reconfigurable devices can be moved *during operation execution*, as shown in the following motivational example.

6.1.1 Motivational Example

Let us consider the unified architectural-level synthesis and placement example presented in Chapter 3. The schedule shown in Figure 3.3a is optimal for the case of fixed virtual modules. However, this schedule can be further improved by taking advantage of the property of dynamic reconfiguration of the digital biochip. Consider the placement in Figure 3.3b. Even though the number of free cells on the microfluidic array at time t is higher than the number of cells in $Mixer_1$, the fragmentation of the space makes the placement of $Mixer_1$ impossible. Hence, the operation has to wait until $t + 4$, in Figure 3.3c, when $Diluter_1$ and $Diluter_2$ finish executing, and there are enough free adjacent cells for accommodating the 2×2 mixing module.

However, this delay can be avoided by "shifting" $Diluter_1$ to another location such that the space fragmentation is minimized. For example, by moving the module three cells to the left as in Figure 6.1b, we can place $Mixer_1$ at time t, obtaining the improved schedule in Figure 6.1a. Shifting is done by changing the activation sequence of the electrodes, such that the droplet is routed to the new position, where it continues moving according to the movement pattern. The moving overhead is equal to the routing time to the new destination. We assume that the time required to route the droplet one cell is 0.01 s (see Section 2.2.1). Under this assumption the routing time required for shifting $Diluter_1$ is 3×0.01 s. As moving a module requires establishing the route that the droplet is taking between the two destinations, the moving overhead will add to the complexity of the post-synthesis routing step.

Although routing times are an order of magnitude faster than operation times, complex routes will lead to overhead. For example, for the colorimetric protein



(a) Dynamic schedule (b) Placement at t (c) $t + 4$

Figure 6.1: Dynamic placement

assay case study (see Figure 2.10), for which we have obtained an application completion time of 179 s on a 13 × 13 array, without considering routing (see Table 6.1), the number of cells reported in [73] as being used for routing is 939. Without considering contention for the routes, this means a routing overhead of 9.39 s. This overhead is for the case when modules are fixed. However, if the location of the modules can be changed unconstrained in every time step, the routing overhead may increase too much. Therefore, constraints have to be imposed on the dynamic placement in order to reduce the routing overhead.

In our placement approach presented in Section 6.1.2, we place the following constraints in order to limit the amount of additional routing caused by dynamic reconfiguration: i) moving a module to a new location is allowed only if there is a route on which the droplet can be transported during the shifting of the device; ii) moves are performed only if defragmentation is required, in order to accommodate a new module; and iii) the routing overhead performed in order to accommodate one device should not exceed a given threshold, $Overhead_{max}$.

6.1.2 Algorithm for Synthesis with Dynamic Devices

We extend the Tabu Search-based algorithm proposed in Chapter 5 to consider that modules are allowed to move during operation execution. The new placement algorithm, which considers the dynamic character of reconfigurable devices is presented in Figure 6.2. The *DynamicPlacement* algorithm takes as input the biochip array consisting of the set of cells $\mathcal{C}$, the current placement $\mathcal{P}$ and the module M_i that has to be placed.

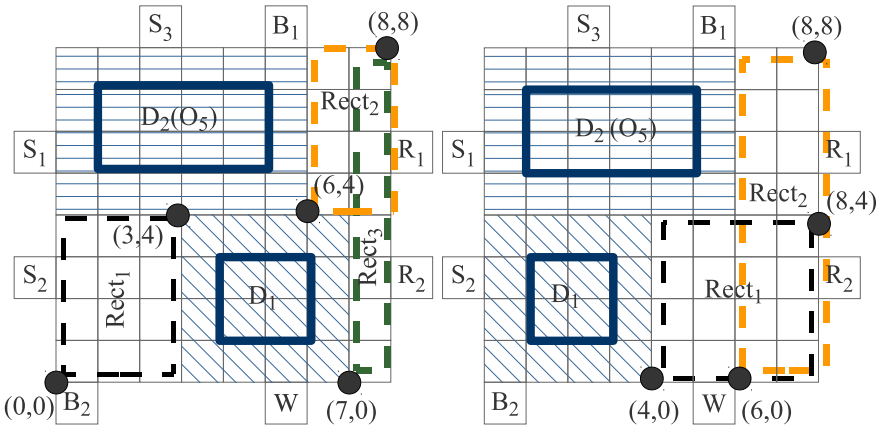
Let us consider the example given in Section 6.1.1 for describing the changes made to the placement algorithm in Figure 5.3. As shown in Figure 6.3a, although O_6 is ready to be executed at time t, the space fragmentation on the microfluidic array makes the placement of the 2 × 2 mixer bound to O_6 impossible. Therefore, in the case of fixed modules the operation has to be delayed until $t + 4$, as shown in Figure 3.3c.

However, when no suitable rectangle can be found for accommodating a device, our modified placement algorithm (presented in Figure 6.2) will try to decrease the space fragmentation on the microfluidic array by moving the modules during their operation (lines 8–28).

We use a greedy approach to decide on which modules to move, until there is space for the current module M_i or a routing time limit is reached. In each iteration of the while loop (lines 12–23) we perform the following steps:

```
DynamicPlacement(C, P, M_i)
1  // construct list of empty rectangles
2  L_rect = ConstructRectList(C)
3  // search for R_i ∈ L_rect that best fits M_i
4  R_i = SelectRectangle(L_rect, M_i)
5  if ∃ R_i then
6    placed = UpdatePlacement(P, R_i, M_i)
7    UpdateFreeSpace(L_rect)
8  else
9    RoutingOverhead = 0
10   MovesList = ∅
11   // dynamically reconfigure already placed modules
12   while ∄ R_i ∧ RoutingOverhead ≤ Overhead_max do
13     R_i = EvaluatePossibleMoves(C, P, L_rect, M_i)
14     if ∃R_i then
15       placed = UpdatePlacement(P, R_i, M_i)
16       UpdateFreeSpace(L_rect)
17     else
18       BestMove = SelectBestMove(C, P, L_rect)
19       PerformMove(BestMove, P, L_rect)
20       RecordMove(MovesList, BestMove)
21       RoutingOverhead = RoutingOverhead + DeterminePerformedRout-
         ing(BestMove)
22     end if
23   end while
24   // no placement has been found, restore the original P
25   if ∄R_i then
26     UndoMoves(P, L_rect, MovesList)
27   end if
28 end if
29 return placed
```

Figure 6.2: Dynamic placement algorithm for DMBs



(a) Initial placement (b) Dynamic reconfigurability

Figure 6.3: Dynamic placement example

1. for each module M_j present on the microfluidic array we evaluate the free space obtained by moving the module in the possible directions (line 13). If one of these moves leads to enough free adjacent space for accommodating the module M_i then the placement is updated by moving M_j in the corresponding direction and placing M_i on the array (lines 14–16).

2. if module M_i has not been placed, the best move evaluated in step 1 is performed (line 19), recorded in a list of moves $MovesList$ (line 20) and the algorithm returns to step 1. The best move during an iteration is considered the one which brings two rectangles as close as possible (minimizing the Manhattan distance[1] between the upper left corners) and at the same time increases the number of free adjacent cells that would be obtained by merging them (line 18). As moving a device requires routing the droplet from the initial position to another one on the array, we place a constraint on the increase in routing time due to moving devices inside the while loop (lines 12–23), of one time step, i.e., $Overhead_{max}$ is one second. Therefore, after each move, the variable $RoutingOverhead$, capturing the extra routing required for moving the droplet between the two locations is updated (line 21). The routing distance is calculated based on the Manhattan distance between the left top corners of the old position and the new position of the module considered for moving. In order to have an accurate approximation of the routing overhead, we consider that a module can be moved only if there are no other modules blocking the

[1]The Manhattan distance between two points with the coordinates (x_1, y_1) and (x_2, y_2), respectively is defined as $d_M = |x_1 - x_2| + |y_1 - y_2|$.

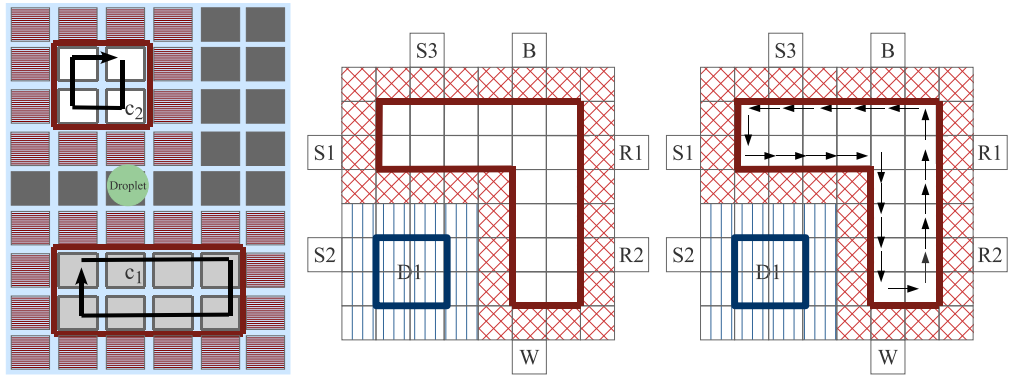
path between the two locations. If not enough free space is thus created for M_i, we restore the previous placement (lines 25–27).

For example, in Figure 6.3a we consider all the possible moves that can be performed on the currently placed modules, $Diluter_1$ and $Diluter_2 \in \mathcal{P}$. As we can see, $Diluter_1$ can be moved at most one cell to the right and three to the left, while $Diluter_2$ can be moved at most two cells to the right. The algorithm will choose to shift $Diluter_1$ three cells to the left, which is the best move: after the move, the Manhattan distance between $Rect_1$ and $Rect_2$ is 6 and the two rectangles contain 24 cells, corresponding to a cost of 30. The existing free space will be updated to $L_{rect} = \{Rect_1 = (4, 0, 8, 4),\ Rect_2 = (6, 0, 8, 8)\}$. As there are now enough adjacent cells, $Mixer_1$ will be placed on the microfluidic array at the bottom of $Rect_1$ and the placement algorithm will terminate.

6.2 Synthesis with Non-Rectangular Devices

In the model presented in Section 6.1 we decreased the space fragmentation on the microfluidic array by dynamically moving the modules while operations are executing. In the second model, we try to further exploit the dynamic reconfigurability of digital biochips by considering changing the shape of the device to which an operation is bound during its execution. As devices are formed by grouping adjacent electrodes, we consider that they can change their size and they can have any shapes, not necessarily rectangular.

Let us consider the example in Figure 6.4a, with the module library shown in Figure 2.1. We assume that 2 s after the mixing operation started executing on the 2 × 4 virtual module, with the droplet being on the cell denoted by c_1, we decide to change the position at which the operation is performed and the number of electrodes used for mixing. In our example, the droplet will be routed to the nearest position belonging to the new group of cells, c_2, where it will continue executing. As the operation was executed for only 2 s out of the 2.9 s required for completion on the 2 × 4 module (see Table 2.1), only 68.96% of the mixing was performed (see Section 2.2.1 on how this percentage is calculated). Next, the operation will continue to execute on the new 2 × 2 group of cells, until the mixing is complete. Considering the completion time of the mixing operation on a 2 × 2 module of 9.95 s as shown in Table 2.1, the remaining 31.04% of the mixing is obtained by routing the droplet inside the 2 × 2 module for 3.08 s. In the end, the overall completion time for the operation is 5.08 s.



(a) Mixing operation (b) L-shaped mixer (c) Movement pattern

Figure 6.4: Execution of a mixing operation

6.2.1 Characterizing Non-Rectangular Modules

Table 2.1 gives completion times for performing reconfigurable operations on various areas. The experiments have considered a limited set of devices, of rectangular shape. However, reconfigurable operations can be executed by routing the droplet on any route, as shown in Figure 6.4b, where a mixing operation is executed on a "L-shaped" virtual module. Since the virtual modules can consist of a varying number of electrodes, arranged in any form, characterizing all devices through experiments is time consuming. Moreover, the completion time of an operation is also influenced by the route taken by the droplet, inside the module, during the execution of the operation. Therefore, we use the analytical method proposed in Section 2.2.1 to determine the completion time of an operation on a module of non-rectangular shape.

For example, for the L-shaped module in Figure 6.4c, routing the droplet once according to the mixing pattern shown by the arrows leads to 8.72% of mixing. Therefore, in order to complete the mixing operation on the L-shaped module, the droplet will be circularly routed on the showned path 11.46 times, leading to a total time of 2.17 s.

6.2.2 Motivational example

Let us consider the graph shown in Figure 6.5. We would like to implement the operations on the 9×9 biochip in Figure 6.6b. We consider the current time as being t. We consider that the input operations are already assigned

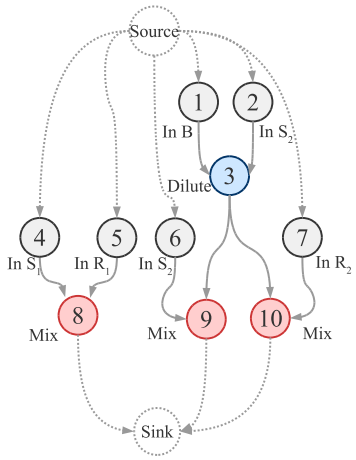


Figure 6.5: Application graph

to the corresponding ports. Thus, O_1 is assigned to the input port B, O_2 to S_2, O_4 to S_1, O_5 to R_1, O_6 to S_2, O_7 to R_2. For simplicity reasons we ignore inputs in this example. However, for the mixing operations (O_8, O_9 and O_{10}) and the dilution operation (O_3) our synthesis approach will have to allocate the appropriate modules, bind operations to them and perform the placement and scheduling.

Let us assume that the available module library is the one captured by Table 2.1. We consider the same execution time for mixing and dilution operations. We have to select modules from the library while trying to minimize the application completion time and place them on the 9×9 chip. A solution to the problem is presented in Figure 6.6b–d, where the following modules are used: one 2×2 mixer ($Mixer_1$), one 2×3 mixer ($Mixer_2$), one 2×4 mixer ($Mixer_3$) and one 1×4 diluter ($Diluter_1$).

Considering the graph in Figure 6.5 with the allocation presented above, Figure 6.6a presents the optimal schedule in the case of static rectangular virtual modules whose locations and shapes remain the same throughout their operation.

Although the schedule presented in Figure 6.6a is optimal for the given allocation and binding, just like in Section 6.1 it can be further improved by reducing the space fragmentation on the microfluidic array. Consider the placement in Figure 6.6c. In order to avoid postponing the execution of O_{10} until $t + 9.95$, we can increase the number of adjacent free cells on the array by changing the location and the shape of the module $Mixer_1$. For example, by re-assigning

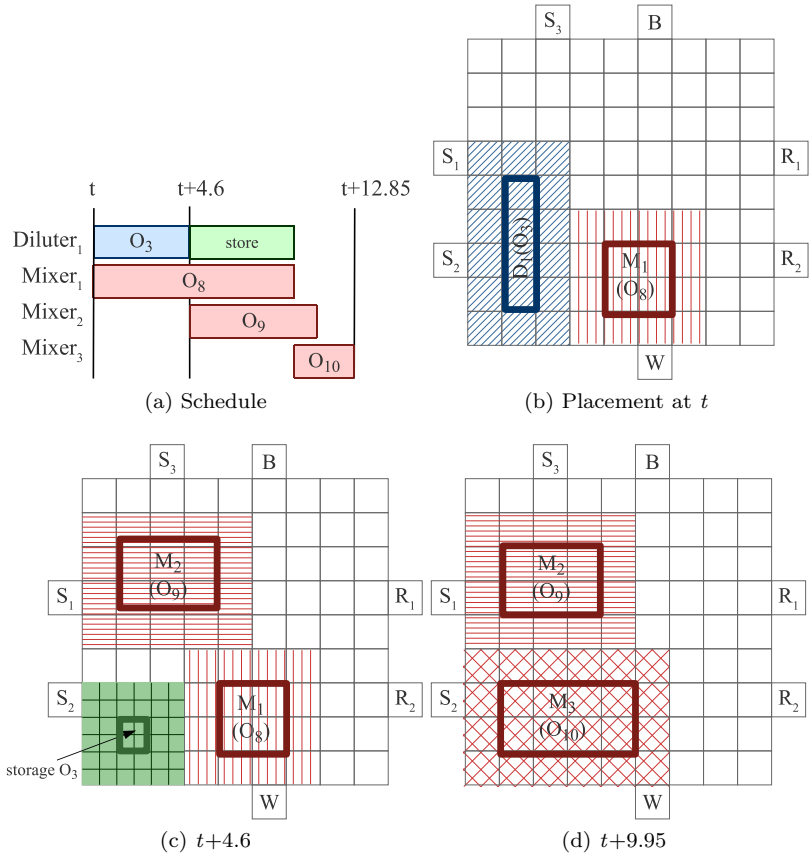
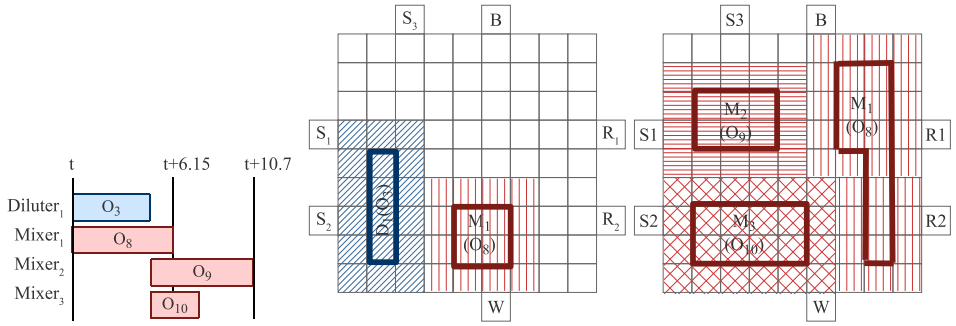


Figure 6.6: Implementation example

the operation to the "L-shaped" device shown in Figure 6.7c and moving the droplet to the new location, we can place $Mixer_3$ at time t+4.6, obtaining the schedule in Figure 6.7a. Shifting is done by changing the activation sequence of the electrodes, such that the droplet is routed to the new position, where it continues moving according to the movement pattern. Considering that at time t+4.6 the mixing operation still had 5.35 s to execute on the 2 × 2 module out of the total 9.95 s, the rest 53.76% of mixing will be executed on the "L-shaped" mixer. Using the method proposed in Section 6.2.1, the completion time of an operation on the "L-shaped" module is 2.89 s, thus the mixing will complete at time $t + 6.15$.



(a) Schedule (b) Placement at t (c) Dynamic reconfigurability at t+4.6

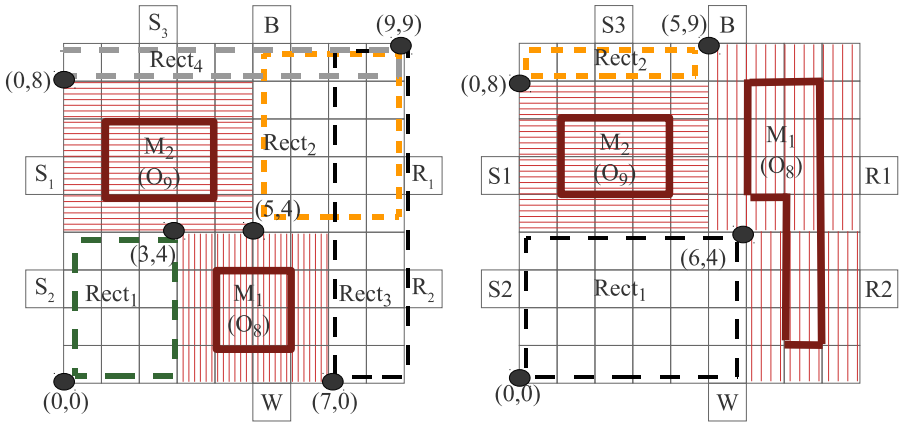
Figure 6.7: Motivational example

6.2.3 Algorithm for Non-Rectangular Modules

Let us use Figure 6.8 to describe the changes to the placement algorithm in Section 5.2 if we consider that modules can dynamically change their shape and location during the operation execution.

Considering the placement in Figure 6.8a, we are trying to decrease the space fragmentation on the array in order to place at the current time the 2×4 module $Mixer_3$, bound to operation O_{10} which is ready for execution. As shown in the figure, $Mixer_1$ can be moved at most three cells to the left and two to the right while $Mixer_2$ can be moved at most four cells to the right and one up. In order to choose the best move we evaluate all moves that can be performed in a greedy fashion: i) we check if the new placement obtained after performing one move while maintaining the initial binding can accommodate $Mixer_3$; ii) if not, we characterize the free space existent on the microfluidic array after the move, considered as a device, and change the shape of the moved device to the new created one; iii) if no space could be created for accommodating $Mixer_3$ we perform the best move possible, the one minimizing the fragmentation of the space. The moving and, if necessary, re-assigning of operations to modules continues until the routing constraint is violated (the routing overhead is exceeded). If not enough adjacent cells have been obtained for placing M_i, the initial placement is restored.

In order to be able to accommodate on the microfluidic array modules of any possible shape, we allow the search engine to group a set of overlapping free rectangles in the case of non-rectangular devices. For example, while evaluating the moves that can be performed on $Mixer_1$ in Figure 6.8a, the algorithm moves



(a) Placing $Mixer_2$ (b) Dynamic reconfigurability

Figure 6.8: Dynamic placement example

$Mixer_1$ two cells to the right. As the move is not sufficient for accommodating $Mixer_3$, we change the module on which $Mixer_1$ is executing. By grouping the free space in the overlapping rectangles $Rect_2 = (5, 4, 9, 9)$ and $Rect_3 = (7, 0, 9, 9)$ we create a new "L-shaped" device, on which $Mixer_1$ can be executed. We assume that the completion time for non-rectangular modules, such as the "L-shape", are computed during the synthesis process, as shown in Section 6.2.1. Once characterized, the devices are added to the given module library for later use. After the re-assignment of $Mixer_1$ to the "L-shape", the free space consists of two rectangles, $Rect_1 = (0, 0, 6, 4)$ and $Rect_2 = (0, 8, 5, 9)$. As there are now enough adjacent cells in $Rect_1$, $Mixer_3$ will be placed on the microfluidic array and the placement algorithm will terminate.

Similar to Section 6.1, we limit the routing overhead performed in order to accommodate a device to a given threshold, $Overhead_{max}$.

6.3 Experimental Evaluation

In this section we present the benefits brought by exploiting the dynamic reconfigurability of operation execution during the synthesis process of DMBs. We have modified the Tabu Search-based algorithm presented in Chapter 5 to consider:

Operation	Area (cells)	Time (s)
Mixing	2 × 5	2
Mixing	2 × 4	3
Mixing	1 × 3	5
Mixing	3 × 3	7
Mixing	2 × 2	10
Dilution	2 × 5	4
Dilution	2 × 4	5
Dilution	1 × 3	7
Dilution	3 × 3	10
Dilution	2 × 2	12
Dispensing	–	7
Detection	1 × 1	30

(a) Module library for the experimental evaluation

Operation	Label	Area (cells)	Time (s)
Mixing	L_1	4 × 2 × 1	1.92
Mixing	L_2	5 × 2 × 1	1.78
Mixing	T	4 × 3 × 1	2.14
Mixing		1 × 5	1.60
Mixing		1 × 6	1.53
Dilution	L_1	4 × 2 × 1	3.78
Dilution	L_2	5 × 2 × 1	3.57
Dilution	T	4 × 3 × 1	4.10
Dilution		1 × 5	3.22
Dilution		1 × 6	3.12

(b) Library of characterized modules

Figure 6.9: Experimental evaluation

- changing the location of modules during operation execution. This is the approach presented in Section 6.1. We denote the corresponding modified Tabu Search implementation by TS^{+}.
- changing the location and shape of modules during their execution, as presented in Section 6.2. The completion time of operations on modules of irregular shapes was computed using the analytical approach described in 2.2.1. We denote the implementation of this approach by TS^{++}.

In order to evaluate the impact of our approaches on the completion time of applications, we use the colorimetric protein assay (see Section 2.4.3) and ten synthetic benchmarks. The module library used for all the experiments is shown in Figure 6.9a. For simplicity, we have considered in the implementation of TS^{++} that the characterization of new modules is done offline. For example, Figure 6.9b contains a set of devices of different shapes, characterized from the given module library in Figure 6.9a. The non-rectangular devices (having "L" and "T" shapes) are described by the lengths of the two segments and the thickness. During the synthesis process, the operations can be re-bound to one of the other devices in Figure 6.9a or to a new device characterized in Figure 6.9b.

For the first set of experiments we were interested in the gains that can be obtained by allowing the dynamic reconfiguration of the devices during their execution. Table 6.1 presents the comparison between the TS implementation for static modules (TS), the one for moving modules (TS^{+}) and the one for non-rectangular modules (TS^{++}) for the protein application (see Section 2.4.3).

As we can see, taking into account the dynamic reconfigurability property of the biochip, significant improvements can be gained in terms of schedule length, allowing us to use smaller areas and thus reduce costs. For example, in the most constrained case, a 11 × 12 array, using the TS^{++} approach, we have obtained an improvement of 10.73% in the average completion time compared with the static modules implementation TS , for the same limit of time, 1 minute.

In a second set of experiments we have evaluated our proposed method on ten synthetic applications. Due to the lack of biochemical application benchmarks, we have generated a set of synthetic graphs using Task Graphs For Free (TGFF) [9]. We have manually modified the graphs in order to capture the characteristics of biochemical applications. The applications are composed of 10 up to 100 operations and the results in Tables 6.2 and 6.3 show the best and the average completion time obtained out of 50 runs of TS and TS^{+}, respectively TS and TS^{++} using a time limit of 10 minutes.

For each synthetic application we have considered three areas, from $Area_1$ (largest) to $Area_3$ (smallest). The results confirm the conclusion from Table 6.1: as the area decreases, considering dynamic reconfiguration becomes more important, and leads to significant improvements. For example, for the synthetic application with 50 operations, in the most constrained case, a 9 × 9 array, using the TS^{+} implementation we have obtained an improvement of 11.18% in the average completion time compared with TS. Moreover, allowing devices to change their shape during execution further increases this improvement to 24.52%.

Area	Time limit	Best			Average			Standard dev.		
		TS^{++}	TS^{+}	TS	TS^{++}	TS^{+}	TS	TS^{++}	TS^{+}	TS
13 × 13	60 min	178.49	179	182	182.03	187.58	189.88	2.53	2.68	2.90
	10 min	178.49	179	182	188.42	187.89	192.00	4.53	3.55	3.64
	1 min	187.49	185	191	194.09	195.13	199.20	4.07	4.27	4.70
12 × 12	60 min	178.49	183	182	183.38	189.76	190.86	3.09	3.01	3.20
	10 min	178.49	185	185	189.99	191.84	197.73	4.41	2.87	6.50
	1 min	190.50	187	193	195.13	206.80	212.62	8.97	7.74	10.97
11 × 12	60 min	178.49	182	184	189.18	191.48	192.50	5.50	3.63	3.78
	10 min	178.49	186	194	193.85	200.40	211.72	4.90	10.20	14.37
	1 min	191.50	204	226	225.13	232.80	252.19	9.27	11.34	15.76

Table 6.1: Comparison between TS^{++}, TS^{+} and TS for the colorimetric protein assay

Nodes	$Area_1$	$Best_1$		$Average_1$		$Area_2$	$Best_2$		$Average_2$		$Area_3$	$Best_3$		$Average_3$	
		TS^+	TS	TS^+	TS		TS^+	TS	TS^+	TS		TS^+	TS	TS^+	TS
10	9×7	20	20	21.94	24.00	7×8	19	20	22.56	25.19	8×6	27	27	29.12	32.58
20	8×7	55	55	55.06	55.16	7×7	58	58	58.53	58.61	6×7	61	67	62.07	67.33
30	10×11	39	41	52.23	56.00	10×10	41	41	55.35	60.78	9×11	46	54	59.60	66.52
40	10×11	56	58	63.82	68.58	10×10	56	58	69.48	76.50	9×10	66	67	76.92	86.37
50	10×10	103	104	107.24	117.78	8×11	111	112	123.70	132.44	9×9	109	119	127.50	143.56
60	11×12	107	110	111.24	113.50	10×11	109	112	112.38	115.40	9×10	112	118	117.94	125.58
70	12×12	121	121	127.69	131.87	11×12	122	123	129.72	137.66	10×11	129	136	143.44	159.72
80	12×12	151	154	159.40	161.60	11×11	165	165	186.54	192.86	10×11	165	168	196.60	210.90
90	15×15	120	120	127.64	128.02	14×14	120	127	131.96	135.68	13×13	133	142	153.86	164.20
100	15×15	163	163	175.04	178.36	14×14	161	170	175.66	179.90	13×13	170	175	175.42	183.84

Table 6.2: Comparison between TS^+ and TS for synthetic benchmarks

Nodes	$Area_1$	$Best_1$ (s)		$Average_1$ (s)		$Area_2$	$Best_2$ (s)		$Average_2$ (s)		$Area_3$	$Best_3$ (s)		$Average_3$ (s)	
		TS^{++}	TS	TS^{++}	TS		TS^{++}	TS	TS^{++}	TS		TS^{++}	TS	TS^{++}	TS
10	9 × 7	15.10	20	20.23	24.00	7 × 8	16.19	20	21.39	25.19	8 × 6	24.20	27	28.60	32.58
20	8 × 7	51.82	55	54.37	55.16	7 × 7	54.12	58	55.72	58.61	6 × 7	60.48	67	63.86	67.33
30	10 × 11	37.20	41	46.47	56.00	10 × 10	37.30	41	53.93	60.78	9 × 11	44.49	54	56.10	66.52
40	10 × 11	49.49	58	53.55	68.58	10 × 10	53.59	58	58.81	76.50	9 × 10	54.59	67	66.25	86.37
50	10 × 10	97.89	104	101.54	117.78	8 × 11	98.97	112	107.85	132.44	9 × 9	99.69	119	108.35	143.56
60	11 × 12	106.69	110	111.56	113.50	10 × 11	106.69	112	111.84	115.40	9 × 10	112.09	118	117.77	125.58
70	12 × 12	119.99	121	123.01	131.87	11 × 12	120.09	123	125.09	137.66	10 × 11	123.39	136	143.43	159.72
80	12 × 12	144.39	154	146.80	161.60	11 × 11	153.12	165	176.23	192.86	10 × 11	155.79	168	187.72	210.90
90	15 × 15	114.79	120	127.72	128.02	14 × 14	120.01	127	129.67	135.68	13 × 13	137.29	142	149.76	164.20
100	15 × 15	157.59	163	165.29	178.36	14 × 14	159.49	170	165.81	179.90	13 × 13	159.69	175	166.68	183.84

Table 6.3: Comparison between TS^{++} and TS for synthetic benchmarks

Chapter 7

Module-Based Synthesis with Droplet-Aware Operation Execution

So far researchers have assumed that during operation execution in module-based synthesis the droplet repeatedly follows the same pattern inside the virtual module, leading to an operation completion time determined through experiments. The actual position of the droplet inside the virtual device has been ignored, considering that all the electrodes forming the device are occupied throughout the operation execution. In order to avoid the accidental merging of droplets, it was considered that a device is surrounded by a 1-cell segregation area (see Figure 7.1a).

In this chapter we consider a *droplet-aware execution of microfluidic operations*, which means that we know the exact position of droplets inside the modules at each time step, and we can control them to avoid accidental merging, if necessary. This allows us to utilize better the chip area, since no segregation cells are needed to separate the modules, and improve the routing step, since now the routes can cross over modules, if needed. Another advantage of droplet-aware operation execution, is that it allows the partial overlapping of modules, which can increase parallelism. However, in this chapter we do not consider module overlapping, which is left for future work.

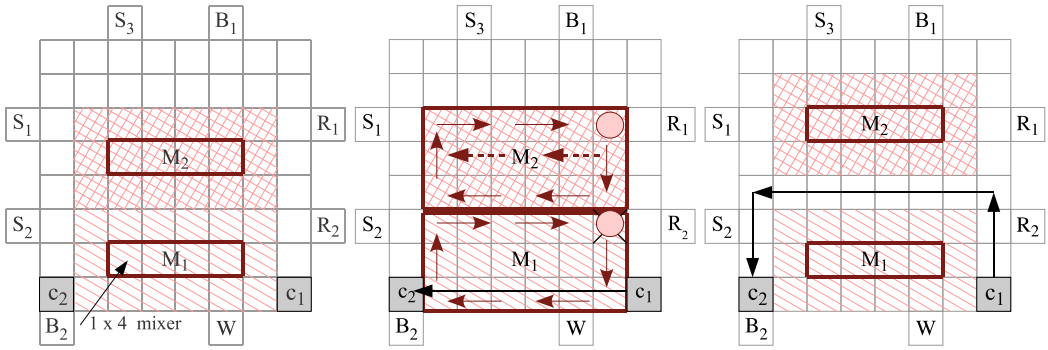


Figure 7.1: Reconfigurable operation execution

Segregation cells have been used so far for ensuring the fluidic constraints during module-based synthesis. However, these cells can be eliminated if we take into account the position of droplets inside modules during operation execution. Let us consider the two mixers in Figure 7.1a. Each mixer is composed of a 1×4 functional area, surrounded by segregation cells to avoid accidental merging. We eliminate the segregation area and consider that the corresponding cells become part of the virtual device (e.g., $Mixer_1$ transforms from a 1×4 to a 3×6 device). We can prevent the accidental merging of the droplets by knowing their locations inside the devices at any time step. For example, considering the initial positions of the two droplets as shown in Figure 7.1b, the mixing operations can be performed by repeatedly routing the droplets according to the movement patterns described by the arrows. The droplets are never too close to each other during execution, so the fluidic constraints are enforced. Such a synchronization of droplets to avoid accidental merging is not always possible.

However, since we know the positions of the droplets we can decide to stop a droplet or change its movement pattern inside a module, to enforce fluidic constraints.

Knowing the locations of droplets inside modules can also be an advantage during the post-synthesis routing step. Let us consider that during the routing step a droplet d must be routed from the cell denoted to c_1 to the cell denoted by c_2 (see Figure 7.1c). The post-synthesis routing algorithms proposed so far have considered devices placed on the chip as obstacles in defining the routes between two modules or between modules and reservoirs, and that the initial placement has to be adjusted in order to introduce the three-cell width paths necessary for

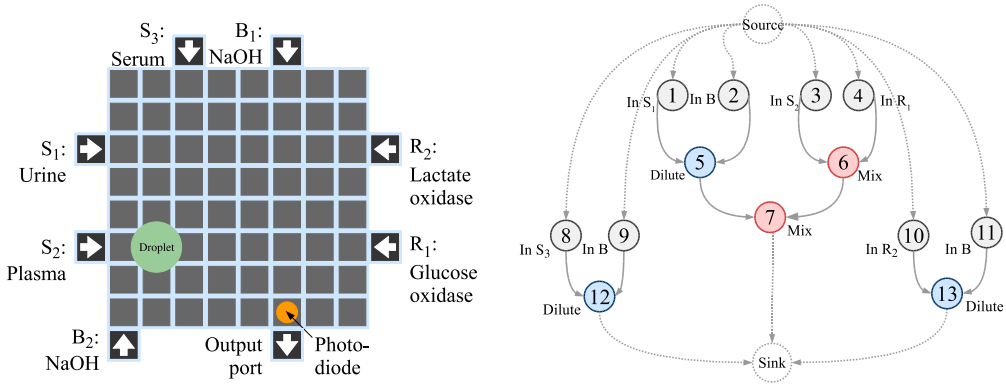
routing, as shown in Figure 7.1c. However, droplets can be routed through the functional area of a module, as long as accidental merging is avoided. Let us assume that at time t the droplets inside the mixers are positioned as shown in Figure 7.1b and are moved according to the pattern shown by the arrows in the mixers. Then droplet d can be routed from the start cell c_1 to the destination cell c_2 on the shortest route possible (shown by the arrow between c_1 and c_2), using electrodes belonging to $Mixer_1$, as long as we ensure the fluidic constraints. For example, in order to avoid the accidental merging inside $Mixer_1$ we can stop the mixer droplet for four time steps on its current position (we mark the stopping place by an "X" on the corresponding electrode). This will allow the routed droplet d to be transported on its optimal path to the electrode denoted by c_2. Due to the fact that the droplets in $Mixer_1$ and $Mixer_2$ are no longer synchronized, we can not continue moving the droplet in $Mixer_2$ according to its original movement pattern, as this would result in an accidental merging with the stopped mixing droplet in $Mixer_1$. Thus, in order to enforce fluidic constraints, we can deviate the movement pattern for the droplet in $Mixer_2$, as shown with dashed arrows in Figure 7.1b.

Changing this movement will result in an irregular pattern, and lead to non-standard operation completion times (i.e., we cannot use numbers such as the ones in Table 2.1, which assume a certain fixed movement pattern). Hence, we use instead the execution time calculation method proposed by us in Section 2.2.1 to compute the completion time of an operation on a droplet-aware device.

The analytical method in Section 2.2.1 takes into account the exact movement pattern of a droplet inside a device to give a safe conservative estimate of the operation completion time. We use the routing approach presented in Chapter 8 to decide the initial location of droplets inside modules.

7.1 Motivational example

Let us consider the graph shown in Figure 7.2b. We would like to implement the operations on a 8×8 biochip shown in Figure 7.2a. We assume that the locations of reservoirs have been decided during the fabrication of the chip and are as shown in Figure 7.2a. We need to assign each input operation to a reservoir of the same type, e.g., O_2 can only be assigned to one of the buffer reservoirs B_1 and B_2. Let us consider that the input operations are assigned to the input ports as follows: O_1 to the input port S_1, O_2 to B_1, O_3 to S_2, O_4 to R_1, O_8 to S_3, O_9 to B_1, O_{10} to R_2 and O_{11} to B_2.



(a) Microfluidic array (b) Application graph

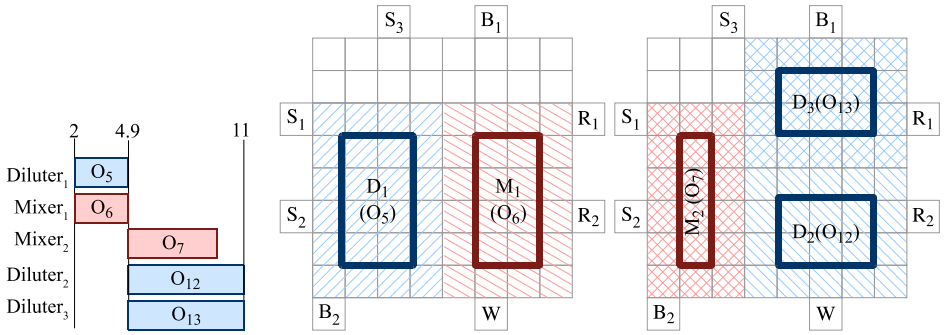
Figure 7.2: Microfluidic array and application graph

Let us assume that the available module library is the one captured by Table 2.1. We consider the same execution time for mixing and dilution operations. We have to select modules from the library while trying to minimize the application completion time and place them on the 8 × 8 chip. We ignore the position of droplets inside modules, and we wrap the modules in segregation cells.

One solution to the problem when considering black-box operation execution is presented in Figure 7.3, where the following modules are used: one 2 × 4 mixer (4 × 6 with segregation area), one 2 × 4 diluter (4 × 6 with segregation area), one 1 × 4 mixer (3 × 6 with segregation area) and two 2 × 3 diluters (4 × 5 with segregation area). The resulted schedule for this allocation is shown in Figure 7.3a.

Considering the graph in Figure 7.2b and the allocation presented above, Figure 7.3a presents the optimal schedule in the case when do not consider the position of droplets inside the virtual modules. We consider that input operations are scheduled for execution as follows: $O_1^{start} = O_2^{start} = O_3^{start} = O_4^{start} = 0$ s, $O_8^{start} = O_9^{start} = O_{10}^{start} = O_{11}^{start} = 2.9$ s. For space reasons, we do not show the schedule of input operations, however the starting times of the reconfigurable operations shown in Figure 7.3a do take into consideration the time required for dispensing the droplets on the microfluidic array. The placement for the allocation and schedule is as indicated in Figures 7.3b–c.

The schedule presented in Figure 7.3a is optimal for the given allocation considering that the positions of droplets inside modules are unknown during operation execution. Therefore, modules are surrounded by segregation cells which ensure



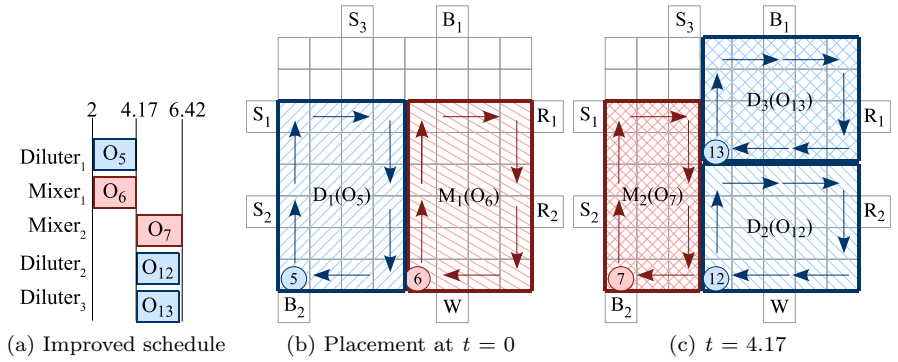
(a) Schedule with segregation cells (b) Placement at $t = 0$ (c) $t = 4.9$

Figure 7.3: Black-box operation execution example

that the fluidic constraints are satisfied at each time step. However, the schedule can be further improved (see Figure 7.4a) by taking into account the location of droplets inside virtual modules. Consider the same synthesis example as in Section 3.1.1, with the allocation presented in Figure 7.3a. At time $t = 2$ operations O_5 and O_6 are scheduled, and modules D_1 and M_1 are placed on the chip. Let us assume that the droplets corresponding to the two operations are routed to the positions shown in the Figure 7.4b, where the dilution and mixing operations start executing, according to the shown movement patterns.

We eliminate the segregation cells, and consider them as part of the functional areas of the devices. For example, operation O_5 who was initially bound to a 2×4 device can now be executed by routing the corresponding droplet on a 4×6 area. The area occupied for performing O_5 remains the same as in Figure 7.3, however, all the cells in the device can now be used for operation execution. By routing the droplets corresponding to O_5 and O_6 as shown in Figure 7.4b, the droplets are never too close and therefore the fluidic constraints are enforced. The same situation is shown in Figure 7.4c, where operations O_7, O_{12} and O_{13} are repeatedly routed from their initial positions, according to the depicted movement patterns, without the need of segregation cells.

The completion times for the droplet-aware operations shown in Figure 7.4 are computed using the analytical method proposed in Section 2.2.1. Although for simplicity reasons the movement patterns of the droplets in Figure 7.4 are synchronized, this is not always possible due to fluidic constraints. Our approach takes this into consideration by allowing a flexible movement pattern of the droplets during operation execution. In order to avoid accidental merging, a droplet can be deviated from its pre-established movement pattern according to



(a) Improved schedule (b) Placement at $t = 0$ (c) $t = 4.17$

Figure 7.4: Droplet-aware operation execution example

the characterized module library (see Figure 7.6d) or can be kept at the same location on the chip for several time steps (see Figure 7.6f).

The exact routes taken by droplets inside a module during operation execution are determined offline and are stored in the memory of a microcontroller, which coordinates the activation of the electrodes on the microfluidic array. In order to minimize memory requirements we consider that only the pre-established routes and the deviations of the droplets from these routes will be recorded in memory, in a compressed form.

7.2 Algorithm for Droplet-Aware Operation Execution

We use the Tabu Search-based algorithm proposed in Chapter 5 to solve the module-based synthesis problem with droplet-aware operation execution. The combined scheduling and placement during operation execution is implemented by the *ScheduleAndPlace* function (Figure 5.2). In order to determine the routing during operation execution, *ScheduleAndPlace* calls the *RunOperationsOneTimeStep* function from Figure 7.5, before the end of each iteration.

The *RunOperationsOneTimeStep* algorithm takes as input the list of operations executing at $t_{current}$, $L_{execute}$, the $m \times n$ matrix $\mathcal{C}$ of cells, the current placement of modules $\mathcal{P}$, the partial routes $\mathcal{R}$ of droplets inside devices up to time $t_{current}$, the module library $\mathcal{L}$, and the current time step $t_{current}$. For each operation O_i under execution at $t_{current}$, the algorithm decides the movement of

RunOperationsOneTimeStep($L_{execute}, \mathcal{C}, \mathcal{P}, \mathcal{R}, \mathcal{L}, t_{current}$**)**

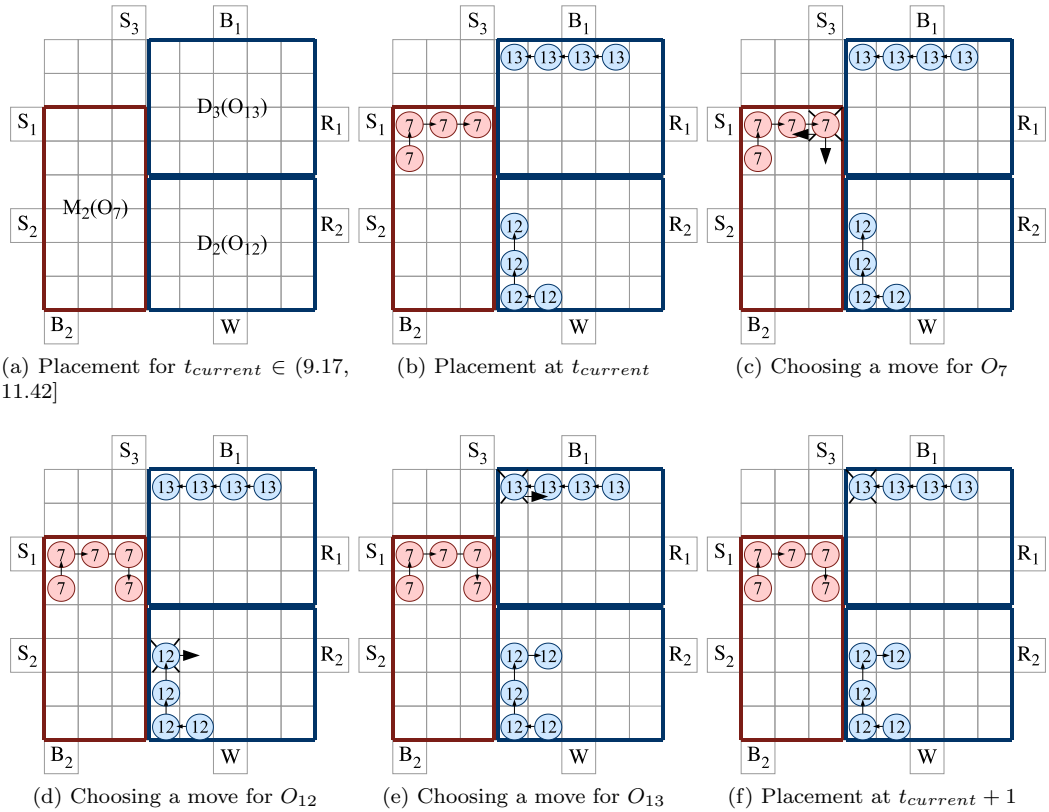
```
1  for all O_i ∈ L_execute do
2    for all direction ∈ {left, right, up, down, stop} do
3      EvaluateMove(C, P, R, L, O_i, direction)
4    end for
5    direction_best = GetBestMove(C, P, R, L, L_execute, O_i)
6    PerformBestMove(O_i, direction_best, C, R)
7    UpdateOperationCompletion(O_i, R_i)
8    if O_i finished executing then
9      t_i^finish = t_current
10   end if
11 end for
12 return R
```

Figure 7.5: Droplet-Aware Operation Execution Algorithm for DMBs

the corresponding droplet inside the module $M_k = \mathcal{B}(O_i)$ the operation is bound to, for the next time step. Compared to previous approaches, we consider that the movement pattern followed by a droplet during operation execution can be dynamically changed, in order to ensure fluidic constraints, and at the same time minimize the completion time of the operation.

The analytical method proposed in Section 2.2.1 is used for characterizing the execution of operations, decomposing the basic modules of a given module library $\mathcal{L}$. According to this method, any route can be decomposed into a sequence of forward, backward and perpendicular moves. In order to determine the completion time of an operation following an irregular movement pattern, we need to approximate the percentage of execution performed over one cell, corresponding to each type of move. The method proposed in Section 2.2.1 provides safe estimates of completion percentages, by decomposing the modules in the given module library, which have pre-established movement patterns and known completion times, determined through experiments. As a result, the method can be used to approximate the amount of operation completion for any given droplet, during operation execution.

Let us consider the example in Figure 7.6a, at time $t_{current}$. There are three operations executing on the array: O_7, bound to a 3×6 mixer module and O_{12} and O_{13}, bound to 4×5 diluters. Let us consider that the previous three moves for the operations are as indicated in Figure 7.6b, by the position of the droplets, and the corresponding connecting arrows. We use a greedy approach for deciding the directions in which the droplets are moved *at the current time step*. For each droplet we consider all the possible directions in which it can



(a) Placement for $t_{current} \in (9.17, 11.42]$ (b) Placement at $t_{current}$ (c) Choosing a move for O_7

(d) Choosing a move for O_{12} (e) Choosing a move for O_{13} (f) Placement at $t_{current} + 1$

Figure 7.6: Running operations O_7, O_{12} and O_{13} for one time step

be moved inside its device, while ensuring that the accidental merging with neighboring droplets is avoided (lines 2–4 in Figure 7.5). The percentages of operation completion gained by performing each of the moves, $\mu_{direction}$, are evaluated using the method from Section 2.2.1. Consequently, according to our greedy approach, the droplet is moved in the direction which leads to the highest percentage μ_{best} (line 6) and the current completion percentage of the operation is updated correspondingly (line 7). If the operation finished executing (i.e, its completion percentage reached 100%) then its finishing time is also updated (lines 8–10).

Let us consider that the first droplet to be moved in Figure 7.6 is the one corresponding to the mixing operation O_7. The droplet can be moved downwards, backwards or it can remain at the current position, see Figure 7.6c. Based on the droplet characterization in Section 2.2.1, the droplet is routed downwards,

as this leads to the most mixing out of the feasible moves. After O_7 is routed, the next droplet to be moved is O_{12}. The droplet cannot continue its movement upwards, as it risks to accidentally merge with O_7 (see Figure 7.6d). Hence, as shown in Figure 7.6e, O_{12} is transported to the right compared to its current position, which is the best possible move. Finally, the algorithm chooses to keep O_{13} on the current position, as moving it backwards leads to negative mixing and moving it downwards breaks the fluidic constraints (accidental mixing with O_7). Figure 7.6f shows the positions of the droplets at time $t_{current} + 1$, after the moves have been performed.

7.3 Experimental Evaluation

In order to evaluate our droplet-aware operation execution approach, we have used two real-life applications and three synthetic TGFF-generated benchmarks. The Tabu Search algorithm was implemented in Java (JDK 1.6), running on Intel Core i7 860 at 2.8 GHz with 8 GB of RAM. The droplet movement characterization of operation execution is based on the decomposition of devices shown in Table 2.1, using the analytical method proposed in Section 2.2.1.

In our experiments we were interested to determine the improvement in completion time that can be obtained by eliminating segregation cells and considering the position of droplets inside devices. Thus, we consider two approaches to the synthesis problem: a droplet-aware operation execution approach (Droplet-Aware Synthesis, DAS) and a black-box operation execution approach, which is the TS approach proposed in Chapter 5.

In order to determine the initial positions of droplets inside modules during droplet-aware operation execution, we have used the GRASP routing method from Chapter 8.

Table 7.1 presents the results obtained by using DAS and TS for the synthesis of two real-life applications: In-vitro diagnostics on human physiological fluids (see Section 2.4.2) and the colorimetric protein assay (see Section 2.4.3). Columns 3 and 4 in the table represent the best solution out of 50 runs (in terms of the application completion time $\delta_{\mathcal{G}}$) for the droplet-aware approach and black-box approach, respectively. The average and standard deviation over the 50 runs compared to the best application completion time are also reported in Table 7.1. The comparison is made for three progressively smaller areas. In Chapter 5 we have shown that the quality of solutions produced by the TS implementation does not degrade significantly if we reduce the time limit from 60 minutes to 10 minutes. Hence, we have decided to use a time limit of 10 minutes for all the

Application	Area	Best		Average		Standard dev.	
		DAS	TS	DAS	TS	DAS	TS
In-vitro	8 × 9	69.83	70.40	72.41	75.72	1.86	3.01
	8 × 8	71.69	82.43	83.67	91.31	11.73	9.63
	7 × 8	74.13	86.82	82.93	95.73	8.01	8.69
Proteins	15 × 15	96.60	102.20	99.66	112.22	1.07	4.63
	14 × 14	95.63	107.12	99.68	116.78	1.12	5.34
	13 × 13	98.76	117.25	101	128.75	0.65	6.46

Table 7.1: Results for the real-life applications

experiments in this chapter. A fast exploration is important since we envision using DAS for architecture exploration, where several biochip architectures have to be quickly evaluated in the early design phase (considering not only different areas, but also different placement of non-reconfigurable resources).

As we can see, controlling the movement of droplets inside devices can lead to improvements in terms of application completion time. For example, in the most constrained case for the colorimetric protein assay (the 13 × 13 array in Table 7.1), we have obtained an improvement of 15.76% in the best schedule length and 21.55% in the average schedule length. Note that the comparison between DAS and TS is unfair towards DAS. In DAS, the completion times presented in the table include routing times (moving the droplets between the devices). There are no routing times in the results reported for TS, where we consider that routing is done as a post-synthesis step, which will introduce additional delays.

A measure of the quality of a Tabu Search implementation is how consistently it produces good quality solutions. The results shown in Table 7.1 were obtained for 50 runs of the DAS and TS approaches. The standard deviations over the 50 runs compared to the best application completion times $\delta_{\mathcal{G}}$ are reported in columns 7 and 8, respectively. As we can notice the standard deviation with DAS is small, which indicated that DAS consistently finds solutions which are very close to the best solution found over the 50 runs (each run will explore differently the solution space, resulting thus in different solutions).

In a second set of experiments, we have compared DAS with TS on three synthetic applications. The graphs are composed of 20, 40 and 60 operations and the results in Table 7.2 show the best and the average completion time, as well as the standard deviation obtained out of 50 runs for DAS and TS, using a time limit of 10 minutes.

Operations	Area	Best		Average		Standard dev.	
		DAS	TS	DAS	TS	DAS	TS
20	8 × 8	40.99	45.01	41.79	47.63	0.8	2.01
	7 × 8	41.32	45.75	43.15	50.46	0.98	2.64
	7 × 7	42.15	47.81	46.23	56.77	1.50	6.14
40	9 × 10	46.85	49.60	47.25	53.93	0.17	2.58
	9 × 9	47.38	51.10	47.76	55.49	0.25	2.60
	8 × 8	47.47	83.83	55.16	92.35	12.27	4.47
60	9 × 10	82.69	84.00	84.88	89.07	1.26	3.11
	9 × 9	82.40	85.43	85.27	95.14	1.40	5.02
	8 × 9	87.54	100.56	95.87	111.89	4.19	7.18

Table 7.2: Results for the synthetic benchmarks

For each synthetic application we have considered three progressively smaller areas. As shown in Table 7.2 the DAS approach leads to significant improvements in the average completion time, compared to the black-box approach. For example we have obtained an improvement of 40.27% in the average schedule length for the application with 40 operations, in the case of the 8 × 8 array.

Chapter 8

Routing-Based Synthesis

In this chapter we remove the concept of "virtual device" and allow operations to execute by routing the droplets on any sequence of electrodes on the array. We call this approach routing-based operation execution.

Similar to the problem formulation for module-based synthesis, during routing-based synthesis we want to synthesize an implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P}, \mathcal{R} >$, deciding the allocation, binding, scheduling, placement and routing. However, there are differences when performing routing-based synthesis. The allocation, binding and placement need to be performed only for non-reconfigurable operations, such as input and detection operations. For reconfigurable operations, such as mixing and dilution, the synthesis is determined by the routes $\mathcal{R}$. For each reconfigurable operation O_i we have to determine a time-ordered list containing electrodes on which O_i is executed (i.e., a route). Thus, for reconfigurable operations, the synthesis problem is transformed into a routing problem.

8.1 Motivational Example

Let us consider the synthesis problem of the application shown in Figure 8.1 on a 7×7 array. We consider the current time step as being $t = 0$. For simplicity,

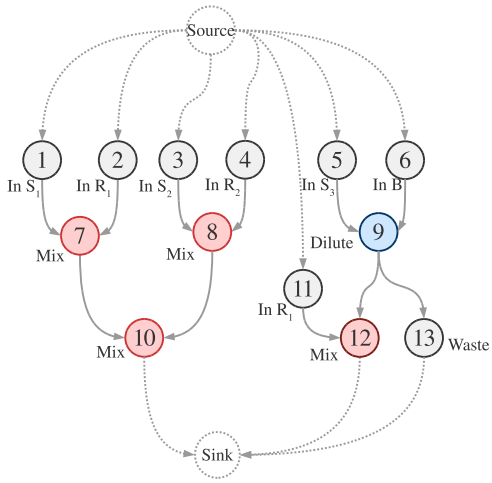
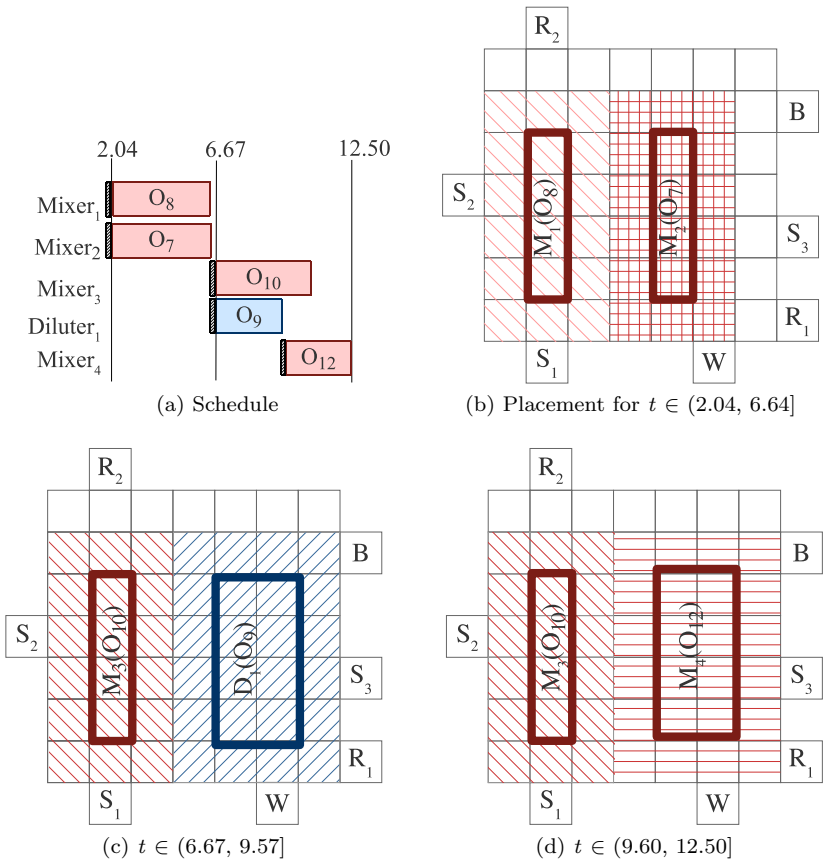


Figure 8.1: Application graph

in this example, we consider that the input operations are already assigned to the corresponding input ports. Thus, O_1 is assigned to the input port S_1, O_2 to R_1, O_3 to S_2, O_4 to R_2, O_5 to S_3, O_6 to B and O_{11} to R_1. Let us assume that the available module library is the one captured by Table 2.1. We consider the same execution time for mixing and dilution operations. We have to select modules from the library while trying to minimize the application completion time and place them on the 7×7 chip.

Let us first consider the case of module-based synthesis, as presented in Chapter 5. The optimal solution to the problem is presented in Figure 8.2, where the following modules are used: three 1×4 mixers ($Mixer_1$, $Mixer_2$, $Mixer_3$), one 2×4 mixer ($Mixer_4$) and one 2×4 diluter ($Diluter_1$). Due to space reasons the schedule presented in Figure 8.2a does not include input operations, however, the starting times for the shown operations consider the time it takes to dispense droplets on the microfluidic array. The routing times needed for merging the inputs of the operations are also included, being represented as hashed rectangles in the schedule. For example, operation O_{12} is bound to module $Mixer_4$, starts after the dilution operation O_9 is completed ($t_9^{finish} = 9.57$) and after its inputs are merged on the microfluidic array, thus $t_{12}^{start} = 9.60$. The operation takes 2.9 s, finishing at time $t_{12}^{finish} = 12.50$ s.

The placement for the solution is as indicated in Figure 8.2b–d. Note that only two virtual devices can be placed on the biochip due to space constraints, thus only two operations can execute in parallel. In our case O_7, O_8 and O_9 could potentially be executed in parallel. If we decide to select smaller areas to increase



(a) Schedule (b) Placement for $t \in (2.04, 6.64]$

(c) $t \in (6.67, 9.57]$ (d) $t \in (9.60, 12.50]$

Figure 8.2: Module-based synthesis example

parallelism, such as a 2×2, the execution time is much larger, e.g., 9.95 s for a 2×2, which eliminates the potential gain obtained through parallelism.

Let us now consider the same problem in the case of routing-based synthesis. We assume the characterization of operation execution as discussed in Section 2.2.1. We have to find the routes $\mathcal{R}$ for all the reconfigurable operations such that the application completion time $\delta_{\mathcal{G}}$ is minimized.

Figure 8.3 shows a complete solution for synthesizing the application $\mathcal{G}$ in Figure 8.1 on a 7×7 chip. Before the reconfigurable operations O_7, O_8 and O_9 can start, we route their inputs to the locations depicted in Figure 8.3b. In order to simplify the visual representation of the solution, we assume a repet-

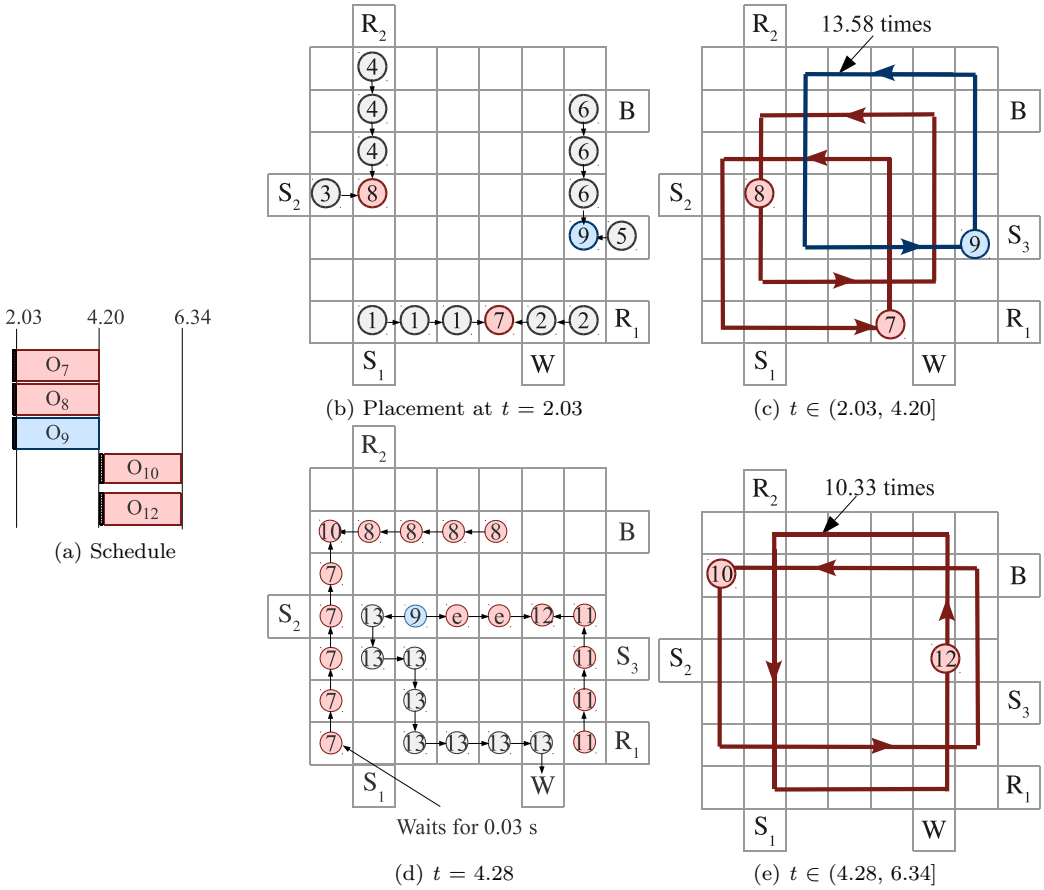


Figure 8.3: Routing-based synthesis example

itive route for the operations: the droplets corresponding to O_7, O_8 and O_9 in Figure 8.3c are repeatedly routed on the shown paths 13.58 times, until the mixing is completed.

After completion, the droplets resulted from the mixing operations O_7 and O_8 are routed to a common location on the chip, where they merge, forming the droplet corresponding to operation O_{10} (Figure 8.3d). The dilution operation O_9 continues by splitting the mixed droplet into two droplets of intermediate concentration and equal volume, corresponding to $e_{9,12}$ and the output operation O_{13}.

Because of simplicity reasons, in this example, the paths on which the droplets are routed while operations are executed are of rectangular shape. However, in routing-based synthesis any sequence of electrodes can be used as a path, as shown in Figure 2.5b.

The schedule of the operations is presented in Figure 8.3a, where we notice that the completion time of the application is significantly reduced compared to the module-based schedule presented in Figure 8.2a, 6.34 s compared to 12.50 s.

There are several reasons for this reduction. Compared to the solution in Figure 8.2, operation O_9 can be executed in parallel with O_7 and O_8 in Figure 8.3c. Routing-based synthesis leads to an increase in parallelism due to a more efficient use of the microfluidic array. In module-based synthesis the entire module area, including the segregation borders, is considered occupied by the operation. In routing-based operation execution we know the actual position of the droplets, therefore all the other cells can be used, as long as the droplets are not too close to each other (i.e., the microfluidic constraints are enforced). For example, in Figure 8.3d the droplet corresponding to O_7 must be kept on the initial position shown from time 2.20 s until time 2.23 s, in order to prevent the accidental merging with the droplet discarded to the output reservoir (corresponding to the operation O_{13}).

Another reason for the reduction of $\delta_{\mathcal{G}}$ is the increase in the number of electrodes used for forward movement. As discussed in Section 2.2, forward movement reduces flow reversibility inside the droplet, leading to a faster completion of the reconfigurable operations, such as mixing and dilution.

8.2 Algorithm for Routing-Based Synthesis

The routing-based synthesis problem is NP-complete [7]. Several methods have been proposed for routing droplets on the microfluidic array during module-based synthesis, see Chapter 4. However, all these methods consider that routing is performed between virtual devices whose position on the microfluidic array is fixed and determined during the placement step, thus the routes have predefined fixed start- and end-points. In addition, the assumption is that the operation is executed within the virtual device. In our routing-based synthesis approach we eliminate the concept of virtual devices and perform operations while routing, and thus there are no fixed start- and end-points for the routes. Also, to guarantee operation completion, we are not interested in minimizing the routes, but we have to construct routes of a given length. Therefore, the existing algorithms are not directly applicable in our context.

The strategy we use is based on Greedy Randomized Adaptive Search Procedure (GRASP) [15] and decides the routes $\mathcal{R}$ taken by droplets during the execution of reconfigurable operations. The allocation, binding and scheduling for non-reconfigurable operations are decided using a greedy approach when these operations are needed by reconfigurable operations.

The proposed algorithm is presented in Figure 8.4 and takes as input the application graph $\mathcal{G}$, the biochip array $\mathcal{C}$ and the percentages of mixing during droplet movement $\mu = \{p_1^0, p_2^0, p^{90}, p^{180}\}$, and produces an implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P}, \mathcal{R} >$, which minimizes the schedule length $\delta_{\mathcal{G}}$.

Let us first discuss the synthesis of routes $\mathcal{R}$ for the reconfigurable operations. At each time t, a set of droplets corresponding to currently executing reconfigurable operations are present on the microfluidic array. A droplet can be in one of the two states: 1) *merge*—when it needs to come into contact with another droplet; and 2) *mix*—when it performs a mixing or dilution operation. For example, the droplets corresponding to operations O_3 and O_4 in Figure 8.3b are in the *merge* state, as they need to be routed to a common location on the array in order to form the droplet corresponding to the operation O_8. Once it is formed, the O_8 droplet is routed on a sequence of electrodes until the mixing operation is completed. Thus, we say that in Figure 8.3c the droplet corresponding to operation O_8 is in the *mix* state.

We use two lists, L_{merge} and L_{mix}, to capture the operations that are performed on the microfluidic array at time t and that are in the *merge* and *mix* states, respectively. L_{merge} is initialized by considering the operations in the graph that are ready to be scheduled for execution (line 4). The list L_{mix} is initially empty (line 5).

The main part of the algorithm is the while loop, lines 6–32, which terminates when all operations have finished. In each iteration, we increment the current time $t_{current}$ (line 31) and perform the following three steps:

1. We decide the new positions of the droplets present on the chip at $t_{current}$, i.e., $O_i \in L_{merge} \cup L_{mix}$ (lines 7–10);

2. In the second step, we introduce droplets on the array in the *mix* state, in case their predecessor droplets have merged on the chip (lines 11–19); and

3. Finally, when the reconfigurable operations have finished executing (the droplets are mixed or diluted), we remember the finishing time (line 22) and put the resulting droplets in the *merge* state (line 29).

RoutingBasedSynthesis($\mathcal{G}$, $\mathcal{C}$, μ)

```
1  t_current = 0
2  t_{O_i}^start = 0, ∀O_i ∈ G
3  t_{O_i}^finish = 0, ∀O_i ∈ G
4  L_merge = ConstructMergeList(G)
5  L_mix = ∅
6  while ∃O_i ∈ G ∧ t_{O_i}^finish = 0 do
7     // Step 1: move droplets present on the array
8     for all O_i ∈ L_merge ∪ L_mix do
9        PerformMove(O_i, C, R)
10    end for
11    // Step 2: if droplet finished merging
12    for all O_i ∈ L_merge ∧ O_i is merged do
13       // update L_merge
14       Remove(O_i, L_merge)
15       // schedule successors
16       ScheduleSuccessors(O_i)
17       // update L_mix
18       Add(O_i, L_mix)
19    end for
20    // Step 3: if droplet finished mixing
21    for all O_i ∈ L_mix ∧ O_i is mixed do
22       t_{O_i}^finish = t_current
23       // update L_mix
24       Remove(O_i, L_mix)
25       if O_i is a dilution operation then
26          ScheduleSuccessors(O_i)
27       end if
28       // update L_merge
29       Add(O_i, L_merge)
30    end for
31    t_current = t_current + 1
32 end while
33 return Ψ
```

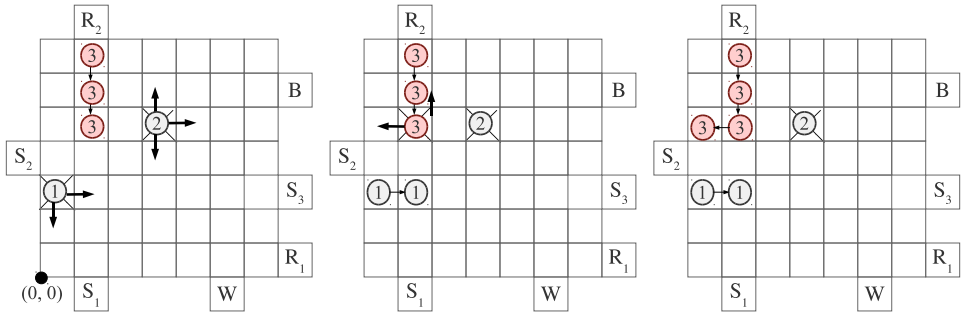
Figure 8.4: Routing-based synthesis for DMBs

Let us present each step in more detail.

1. In step 1, for each droplet present on the microfluidic array, we have to decide the next position (line 9). There is a large number of position combinations that has to be considered. We take the decision individually for each droplet, using the PerformMove function which takes as input the reconfigurable operation O_i, the biochip array $\mathcal{C}$ and the current routes $\mathcal{R}$. We use a randomized greedy approach similar to GRASP: for each droplet we construct a Restricted Candidate List (RCL), containing the three best feasible moves to be performed. Then, a move from the RCL is randomly selected and the droplet is transported in the corresponding direction. We use probabilities to favour the candidates from RCL which have a greater cost function. Thus, there is a probability of 50% of choosing the best candidate from the RCL, 33.3% of choosing the second best candidate and 16.6% for the third best feasible move. Two cost functions are considered for determining the quality of the moves, depending on the state of the droplets. For a droplet in the *mix* state, the quality of a move is given by the percentage of mixing performed while transporting the droplet in the given direction calculated based on the set μ, see Section 2.2.1. For a droplet in the *merge* state, the quality of a move is determined by the distance between the two droplets that need to be merged, measured by the Manhattan distance.

   Let us use Figure 8.5a to illustrate how we determine the directions in which the droplets are moved. We consider that at time $t_{current}$ there are three operations executing on the array: the operations O_1 and O_2, that need to be merged, and the mixing operation O_3, thus $L_{merge} = \{O_1, O_2\}$ and $L_{mix} = \{O_3\}$. As discussed in Section 2.2.1, for an operation executing by routing, the amount of mixing performed during one move depends on the previous path on which the droplet was transported. The previous two moves for mixing operation O_3 are as indicated in Figure 8.5a, by the position of the O_3 droplet, and the corresponding connecting arrows. For each of the droplets on the array we have a number of feasible moves that can be performed at the current time step. In Figure 8.5a the feasible moves are depicted with thick arrows. The set of feasible moves includes also the decision of keeping the droplet on the same position, illustrated with an "X" under the droplet. When considering the feasible moves set we enforce the microfluidic constraints, which prevent the droplets from getting too close to each other and accidentally merge. For example the move of droplet O_1 up is not permitted, since doing so would cause it to merge with droplet O_3.

   The operations in L_{merge} are considered first. For the droplet corresponding to operation O_1 we have three possible moves: to the right, down or maintaining the droplet at the current location. We evaluate each of



(a) Choosing moves for L_{merge} (b) Choosing a move for L_{mix} (c) Placement at $t_{current} + 1$

Figure 8.5: Performing droplet moves

the possible directions, by computing the Manhattan distance between the new feasible position of O_1 and the position of the droplet that O_1 has to merge with, O_2. The current positions of the O_1 and O_2 droplets are (0,2) and (3,4) respectively, thus the initial Manhattan distance is 5, as shown in Figure 8.5a. By moving O_1 to the right the new location of the droplet is (1,2), therefore the Manhattan distance between O_1 and O_2 is reduced to 4. Similarly, the Manhattan distance obtained by moving the droplet O_1 down and maintaining it at the current location are 6 and 5, respectively. Thus, moving O_1 to the right is the best decision, as it brings the droplets O_1 and O_2 closer to each other. The RCL is constructed by considering only the three best moves, thus $RCL_{O_1} = \{right, maintain, down\}$. A move is randomly chosen from the RCL and the placement of the droplet on the chip is updated. Let us consider for example that the droplet is moved to the right. Similarly we construct $RCL_{O_2} = \{down, maintain, right\}$ and randomly choose to maintain the droplet corresponding to O_2 at the current location. Figure 8.5b shows the updated placement on the microfluidic array after the two moves are performed.

Next, the mixing operation O_3 is considered. The feasible directions in which the droplet can be routed are to the left, up or maintaining the droplet on the current position. Moving the droplet in a forward direction is not possible, as this could lead to an accidental merge with the droplet corresponding to O_1 (see Figure 8.5b). As moving the droplet to the left would result in a perpendicular move compared to the previous one, the percentage of obtained mixing according to Section 2.2.1 is p^{90} = 0.10%, while moving it backwards (up) would result in a negative mixing, $p^{180} = -0.5\%$. We consider mixing by pure diffusion negligible, thus no mixing is performed while the droplet remains at the same location.

Therefore, $RCL_{O_3} = \{left, maintain, up\}$. We assume the droplet is randomly moved to the left, resulting in the placement shown in Figure 8.5c.

2. In Step 2, for all the droplets in L_{merge} that have been brought to a common location at time $t_{current}$, their successors are activated and inserted into the corresponding lists. Their t_{start} is set to $t_{current}$(line 16) and their positions are at the same location where the droplets have met. For example, when O_1 and O_2 in Figure 8.3b are merged at time $t = 2.03$, the mixing operation O_7 is placed on the array and starts executing, thus $t_{O_7}^{start} = 2.03$.

3. In Step 3, all the mixing operations completed at time $t_{current}$ and having successors are promoted to the *merge* state (lines 20–30). For example, at time $t = 4.20$, the state of the operation O_7 in Figure 8.3d is changed from *mix* to *merge*, as O_7 needs to be merged with O_8 in order to form the droplet corresponding to the operation O_{10}. If the completed operation is of type dilution, then the droplet is split into two droplets of equal volumes, see the dilution operation O_9 in Figure 8.3d. The droplets resulting from the split operation are scheduled (line 26) and their locations on the array are determined by their predecessor's final position.

Regarding non-reconfigurable operations, such as dispensing from input reservoirs and detection using optical devices, we consider that their allocation $\mathcal{A}$ and placement $\mathcal{P}$ are fixed and given as part of the biochip architecture model. However, we decide the binding $\mathcal{B} \in \Psi$ and scheduling $\mathcal{S} \in \Psi$ of non-reconfigurable devices as part of the synthesis process. Thus, if a droplet corresponding to an input operation is needed on the microfluidic array at $t_{current}$, we schedule the dispensing of the droplet such that it finishes at time $t_{current}$, and not earlier. This is in order to avoid storing the dispensed droplets on the array, until they are needed by other operations, as they will otherwise occupy space required for performing other operations. Because of the constraint on the number of available reservoirs on a given chip, creating a dispensed droplet at $t_{current}$ is not always possible. In this case, the input operation is bound using a greedy approach to the reservoir that will be available at the earliest time. We use the same approach for determining the binding of detection operations to optical devices.

Due to its randomized nature, the algorithm in Figure 8.4 might produce different results for different runs, with the same inputs. The algorithm terminates when all operations have been synthesized, and returns the solution Ψ (line 33). Our route-based synthesis approach is given a time limit, and runs repeatedly RoutingBasedSynthesis from Figure 8.4 until the time limit is reached, collecting the best solution Ψ in terms of the application completion time $\delta_{\mathcal{G}}$.

8.3 Routing-Based Synthesis with Contamination Avoidance

The synthesis approaches we have proposed so far do not address the problem of cross-contamination of samples during the biochemical application execution. However, as discussed in Chapter 2, some biochemical applications contain liquids that adsorb on the substrate on which they are transported. Consequently, the purity of the droplets routed on the microfluidic array can be affected by the contaminated electrodes and this may lead to an erroneous outcome of the performed biochemical assay. Even though the use of silicon oil minimizes the risk of surface fouling, the complete avoidance of cross-contamination of samples becomes a key challenge when performing critical biochemical applications. Wash droplets are typically used in such cases to clean contaminated sites on the chip, between successive transportations of droplets.

Contamination avoidance increases the complexity of the synthesis problem due to the following reasons:

- additional wash droplets must be scheduled and transported on the contaminated sites on the chip in order to remove the existent residue.

- the flexibility of droplet movement on the microfluidic array is reduced, as contaminated electrodes cannot be used as part of droplets routes.

Several techniques (see Chapter 4) have been proposed so far for contamination avoidance during the synthesis problem of direct addressing biochips. All these methods consider that reconfigurable operations are performed inside modules, ignoring the positions of droplets during operation execution.

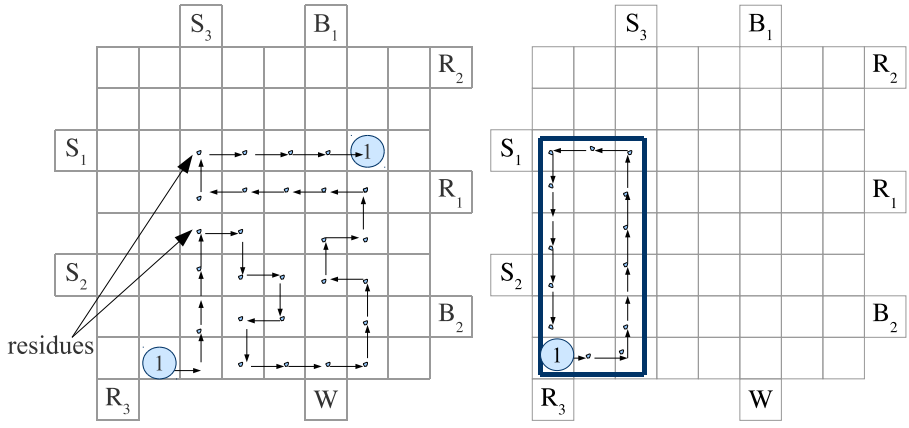
In this and the next sections we present two approaches for contamination avoidance during routing-based synthesis. Since in routing-based operation execution we consider that operations are performed by transporting the corresponding droplets on any sequence of electrodes on the microfluidic array, the potential to contaminate is quite large. The first proposed method, presented in this section, is based on extending the GRASP algorithm from Section 8.2.

8.3.1 Contamination Avoidance in Routing- vs. Module-Based Synthesis

The main advantage of routing-based operation execution is the increase in parallelism, since the same electrode can be used in the routing paths of several executing operations. However, this flexibility in droplet movement can become a disadvantage when contamination is a concern.

Let us consider the example in Figure 8.6a, where a dilution operation O_1 is performed on the microfluidic array by routing the corresponding droplet. If the sequence of electrodes on which the operation is executed is unconstrained, the droplet can be moved on the chip on a free pattern, such as the one shown in Figure 8.6a. However, if substances composing the droplet are adsorbed on the surface of the microfluidic array, the unrestricted route of the droplet can lead to a high number of contaminated electrodes, as shown in Figure 8.6a. In this case it is important that the contaminated sites are cleaned as soon as possible, in order not the block the execution of other operations that are performed concurrently on the chip.

Consider the same example in the case when the execution of the operation is constrained to a specific area. We assume that the dilution operation O_1 is performed by routing the droplet inside a 3×6 area, such as the one shown in Figure 8.6b. Since the droplet is repeatedly transported over the same sequence of electrodes, the contamination is limited to the module area.



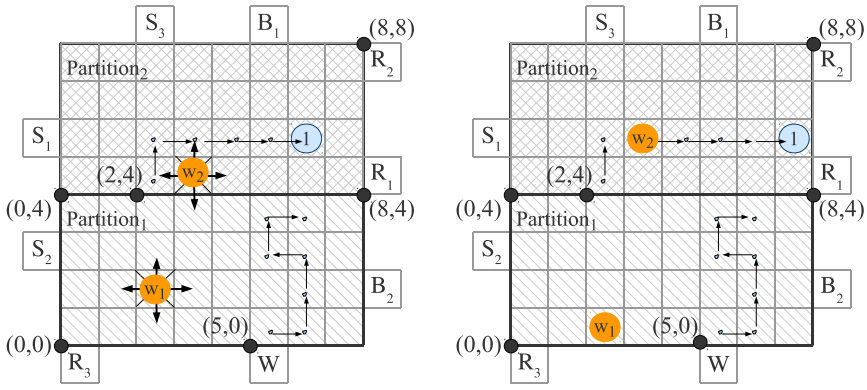
(a) Routing-based operation execution (b) Area-constrained operation execution

Figure 8.6: Contamination during operation execution

The advantage of this approach is the reduced number of electrodes contaminated during the execution of the operation. Moreover, if the areas on which operations are performed on the chip are not overlapping, it is only necessary to clean the contaminated electrodes after the operations performed on them have finished executing.

8.3.2 Algorithm for Routing-Based Synthesis with Contamination Avoidance

This section presents the algorithm for contamination aware routing-based synthesis. The method is based on the partitioning of the chip in a number n_{part} of smaller, equal areas. For example, the microfluidic array shown in Figure 8.7a is divided into two equal partitions: $Partition_1$, represented by the rectangle (0, 0, 8, 4), and $Partition_2$, represented by (0, 4, 8, 8). Each partition on the chip is assigned a wash droplet, denoted in Figure 8.7a by w_1 and w_2, respectively, responsible of cleaning the contaminated electrodes inside the corresponding area. For example, considering the microfluidic array in Figure 8.7a, w_1 is responsible of cleaning the set of electrodes {(5, 0), (6, 0), (6, 1), (6, 2), (5, 2), (5, 3), (6, 3)} and w_2 for the set {(2, 4), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5)}. Similar to the previous related work, we consider that the purity of a wash droplet is reduced with the number of cleaned electrodes. We denote by $max_{electrodes}$ the maximum number of contaminated electrodes that can be cleaned by a wash droplet. We consider that $max_{electrodes}$ will be given as an input in the design specifications.



(a) Microfluidic array at $t_{current}$ (b) Microfluidic array at $t_{current} + 1$

Figure 8.7: Routing-based operation execution with contamination avoidance

We extend the GRASP-based algorithm proposed in Section 8.2 to take into account the contamination problem during routing-based synthesis. The algorithm is shown in Figure 8.8. The input consists of the application graph $\mathcal{G}$, the biochip array $\mathcal{C}$, the module library $\mathcal{L}$, the maximum number of electrodes $max_{electrodes}$ that a wash droplet can clean, and the number of partitions no_{part}, and the output is an implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P}, \mathcal{R} >$, which minimizes the schedule length $\delta_{\mathcal{G}}$, such that the contamination is avoided.

The algorithm starts with the partitioning of the microfluidic array in a number no_{part} of smaller areas. A list L_{wash} is used to keep track of the wash droplets on the array at time $t_{current}$. The list is initialized in line 2 of the algorithm, by creating a number of wash droplets equal to the number of partitions on the chip. Each droplet is assigned a partition and a maximum number of electrodes that it can clean, equal to $max_{electrodes}$. The characterization of droplet movement is performed by decomposing the given module library $\mathcal{L}$ into the set of operation execution percentages μ, based on the analytical method proposed in Section 2.2.1 (line 3).

Compared to the algorithm presented in Section 8.2 we introduce a new state in which a droplet on the microfluidic array can be at time $t_{current}$: the *wash* state. This state is reserved for wash operations that are active on the microfluidic array (i.e., they can still clean contaminated electrodes). The routing of droplets in *merge* and *mix* state is done as explained in Section 8.2. However, we consider that if a droplet has the potential of contaminating the surface of the chip, all electrodes on which it is routed will be marked as contaminated (lines 12–14 in Figure 8.8). These electrodes are then assigned to be cleaned by the droplet attached to the partition the electrodes belong to (line 13). Consider the example in Figure 8.7a. We assume that at time $t_{current}$ the dilution operation O_1 is moved to the right compared to its previous position, as shown in Figure 8.7b. Since the new contaminated electrode at location (7, 5) belongs to $Partition_2$, it will be added to the list of sites to be cleaned by wash droplet w_2.

At each time step our algorithm decides the new locations for all the droplets present on the chip. We integrate the decision for operations in the *wash* state in our GRASP approach presented in Section 8.2. Let us consider the wash droplet w_1 in Figure 8.7a. The droplet can be moved to the left, to the right, upwards, downwards or it can be maintained on the same electrode compared to its current position. For a droplet in the *wash* state the quality of the move is given by the Manhattan distance between the new feasible location of the droplet and the location of the first electrode to be cleaned. For example, for w_1 the best move to be performed is to the right or downwards, as it brings droplet w_1 closer to the first electrode to be cleaned, at location (5, 0). By evaluating all the feasible moves the candidate list RCL_{w_1} = {right, down, maintain} is constructed, containing the best three moves for wash droplet w_1. According to GRASP,

ContaminationAwareRBS($\mathcal{G}, \mathcal{C}, \mathcal{L}, max_{electrodes}, no_{part}$)

```
1  PartitionChip(C, no_part)
2  L_wash = ConstructWashList(max_electrodes, no_part)
3  μ = CharacterizeMovement(L)
4  t_current = 0
5  t_{O_i}^start = 0 ∀O_i ∈ G
6  t_{O_i}^finish = 0, ∀O_i ∈ G
7  L_merge = ConstructMergeList(G)
8  L_mix = ∅
9  while ∃O_i ∈ G ∧ t_{O_i}^finish = 0 do
10   for all O_i ∈ L_merge ∪ L_mix do
11     R_i = PerformMove(O_i, C, R)
12     if O_i contaminates then
13       SetElectrodeContaminated(O_i, R_i, L_wash)
14     end if
15   end for
16   for all O_i ∈ L_merge ∧ O_i is merged do
17     Remove(O_i, L_merge)
18     ScheduleSuccessors(O_i)
19     Add(O_i, L_mix)
20   end for
21   for all O_i ∈ L_mix ∧ O_i is mixed do
22     t_{O_i}^finish = t_current
23     Remove(O_i, L_mix)
24     if O_i is a dilution operation then
25       ScheduleSuccessors(O_i)
26     end if
27     Add(O_i, L_merge)
28   end for
29   for all O_i ∈ L_wash do
30     R_i = PerformMove(O_i, C, R)
31     if R_i is contaminated then
32       SetElectrodeCleaned(R_i, L_wash)
33       UpdateWashCapabilities(O_i)
34       if O_i can not clean anymore then
35         RemoveFromWashList(O_i, L_wash)
36         CreateWashDroplet(O_i, L_wash)
37       end if
38     end if
39   end for
40   t_current = t_current + 1
41 end while
42 return Ψ
```

Figure 8.8: Contamination-aware routing-based synthesis for DMBs

a move is randomly chosen from RCL_{w_1} and the droplet is transported in the corresponding direction. Let us assume that the droplet is moved downwards, as shown in Figure 8.7b. Similarly, the ready candidate list for the wash droplet w_2 is constructed $RCL_{w_2} = \{\text{left, maintain, upwards}\}$. Let us consider that the droplet is transported upwards, see Figure 8.7b. Because the electrode (3, 5) on which the wash droplet has been moved is contaminated, the site is marked as cleaned (line 32) and the wash capabilities of w_2 are updated (line 33). If the wash droplet has reached the limit of $max_{electrodes}$ cleaned electrodes, we consider that its purity has decreased to a point where it can not be used anymore. Therefore, the droplet is sent to the waste reservoir and another wash droplet is dispensed and assigned to the corresponding partition (lines 35–36).

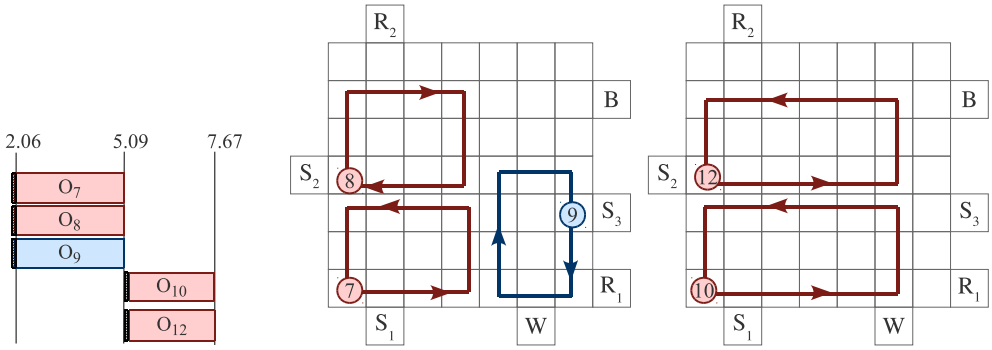
The algorithm terminates when all the operations in the biochemical applications have finished executing.

8.4 Area-Constrained Routing for Contamination Avoidance

In the algorithm presented in the previous section we have considered that operations are executed by transporting the corresponding droplets on any route on the microfluidic array. However, by constraining the execution of operations to a certain area on the microfluidic array, the contamination can be decreased. Therefore, in this section we present an approach in which droplet routes are constrained to a given area during operation execution.

Consider the example presented in Section 8.1 where the mixing operations O_7, O_8, O_{10} and O_{12} and the dilution operation O_9 must be performed on the microfluidic array. We consider that the movement of a droplet in the *merge* state is decided by the GRASP algorithm, as presented in Section 8.2. However, for a droplet in the "mix" state, we constrain the route to a certain area on the microfluidic array, as shown in Figure 8.9. We use the Tabu Search-based algorithm presented in Figure 7.5 to determine for each executing operation the area in which it will be performed and the route of the droplet during execution.

For example, in Figure 8.9 the operations are performed by transporting the droplets as follows: O_7, O_8 and O_9 inside 3×4 modules, O_{10} and O_{12} inside 3×6 modules. The routes during operation execution are decided in a greedy fashion, using the approach proposed in Section 7.2. The analytical method presented in Section 2.2.1 is used to determine the execution time of an operation inside the area where it is executed. In our example, considering the shown movement patterns, Figure 8.9a presents the schedule obtained for area-



(a) Schedule (b) $t \in (2.06, 5.09]$ (c) $t \in (5.18, 7.67]$

Figure 8.9: Area-constrained operation execution

constrained routing. The schedule includes the routing times of operations in the *merge* state, determined using the GRASP algorithm.

8.5 Experimental Evaluation

In order to evaluate our routing-based proposed approach proposed in Section 8.2, we have used two-real life examples and ten synthetic benchmarks. The GRASP-derived algorithm was implemented in Java (JDK 1.6), running on SunFire v440 computers with UltraSPARC IIIi CPUs at 1,062 GHz and 8 GB of RAM. The module library used for all the experiments is shown in Table 2.1. For the following set of experiments we do not consider contamination avoidance.

In our first experiments we were interested to determine the improvement that can be obtained by using Routing-Based Synthesis (RBS) compared to the module-based synthesis approach (MBS) using the Tabu Search from Chapter 5.

Table 8.1 presents the results obtained by using RBS and MBS for two real-life applications: 1) *In-vitro* diagnosis on human physiological fluids (IVD) (see Section 2.4.2), and 2) the *colorimetric protein assay* (see Section 2.4.3). Table 8.1 presents the best solution (in terms of the application completion time $\delta_{\mathcal{G}}$), in columns 3 and 4. The comparison is made for three progressively smaller areas for both approaches, using a time limit of 10 minutes for both synthesis approaches.

As we can see, eliminating the concept of "virtual modules" and allowing the operations to perform on any route on the microfluidic array can lead to significant improvements in terms of application completion time, allowing us to use smaller areas and thus reduce costs. Using routing-based synthesis is particularly important for more constrained synthesis problems, when knowing the exact location of all droplets on the array, leads to more efficient space usage. For example, in the most constrained case for the colorimetric protein assay, the 10×10 array, we have obtained an improvement of 44.95% in the schedule length.

Moreover, the routing-based approach determines a complete solution for the problem, while for the module-based synthesis a post-synthesis step is necessary to determine the routing, which means additional delays.

Both RBS and MBS implementations are stochastic: random decisions during the exploration process can lead to slightly different results. To determine the quality of the RBS implementation, we have run RBS and the Tabu Search-based MBS 50 times. The best results for RBS and MBS, presented in columns 3 and 4 in Table 8.1, respectively, are collected after 50 runs. The average and standard deviation over the 50 runs compared to the best application completion time $\delta_{\mathcal{G}}$ are also reported in Table 8.1. As we can see, the difference between RBS and MBS is larger in the average case, and the standard deviation with RBS is very small, which means that RBS consistently finds solutions which are very close to the best solution found over the 50 runs.

In a second set of experiments we have compared RBS with MBS on ten synthetic applications, generated using TGFF. The graphs are composed of 10 up to 100 operations and the results in Table 8.2 show the best and the average completion time obtained out of 50 runs for RBS and MBS, using a time limit of 10 minutes.

Application	Area	Best		Average		Standard dev.	
		RBS	MBS	RBS	MBS	RBS	MBS
In-vitro (28 operations)	8×9	68.43	72.94	68.77	77.81	0.16	2.12
	8×8	68.87	82.12	69.13	102.37	0.14	13.58
	7×8	69.12	87.33	69.46	111.18	0.17	12.26
Proteins (103 operations)	11×11	113.63	184.06	117.51	205.30	4.65	8.38
	11×10	114.33	185.91	119.62	202.14	6.63	8.84
	10×10	115.65	208.90	120.65	219.17	7.73	7.89

Table 8.1: Results for the real-life applications

Operations	$Area_1$	$Best_1$		$Average_1$		$Area_2$	$Best_2$		$Average_2$		$Area_3$	$Best_3$		$Average_3$	
		RBS	MBS	RBS	MBS		RBS	MBS	RBS	MBS		RBS	MBS	RBS	MBS
10	6×6	39.12	42.61	39.92	42.61	5×7	39.55	76.1	39.95	76.1	5×6	40.46	102.9	40.97	102.9
20	8×8	49.73	52.71	50.18	52.71	7×8	50.5	49.01	50.95	53.62	7×7	51.19	49.81	51.74	60.06
30	8×8	64.73	67	65.96	72.84	7×8	66.92	76.4	67.79	84.08	7×7	68.42	82.49	69.68	95.54
40	8×8	61.18	91.97	61.93	102.69	7×8	63.01	98.25	63.74	111.47	7×7	64.75	99.29	65.85	131.63
50	9×10	83.27	82.4	83.89	86.99	9×9	84.02	87.21	84.76	93.5	8×9	85.37	87.03	86.34	101.59
60	9×9	93.82	89.90	94.98	100.44	8×10	94.34	95.70	95.15	104.80	8×9	94.39	106.7	95.85	122.42
70	10×10	140.4	153.8	179.97	194.91	9×11	155.93	164.01	197.05	182.99	9×10	147.39	162.41	186.02	233.57
80	10×10	112.38	113.4	112.98	124.98	9×10	112.43	124.75	113.48	139.26	9×9	113.6	133.87	114.23	147.86
90	11×11	128.08	127.41	139.33	180.64	10×10	131.32	149.68	144.23	215.76	9×10	136.94	156.31	148.59	227.02
100	11×11	153.06	285.05	172.15	325.57	10×10	154.09	255.97	172.46	321.87	9×11	153.08	278.63	170.17	325.66

Table 8.2: Results for synthetic benchmarks

For each synthetic application we have considered three progressively smaller areas. The results in Table 8.2 confirm the conclusion from Table 8.1: as the area decreases, performing routing-based synthesis becomes more important, and leads to significant improvements. For example, for the synthetic application with 100 operations, in the case of the 9×11 array, we have obtained an improvement of 47.74% in the average completion time compared with module-based synthesis.

In the previous set of experiments we have considered that droplets do not contaminate the surface of the microfluidic array during their transportation. As shown from the results, routing the droplets on any route during operation execution leads to significant improvements, due to a better utilization of the microfluidic array. However, as discussed in Chapter 2, some biochemical applications contain liquids that are adsorbed onto the surface of the chip, leading to a possible contamination of the droplets. In such cases contamination avoidance must be ensured, to provide correct outcomes for the performed applications. Thus, in the next experiments we consider the case of cross-contamination avoidance during the synthesis problem. For this, we evaluate our GRASP algorithm[1] presented in Section 8.3.2 on one real-life application and three synthetic TGFF-generated benchmarks.

In our experiments we were interested to determine the suitability of routing-based synthesis when contamination is a concern. Therefore, we have considered two approaches to the synthesis problem with cross-contamination avoidance: a routing-based synthesis in which droplets are moved freely during operation execution (Routing Based Synthesis with Contamination avoidance, RBSC, presented in Section 8.3.2) and an area-constrained routing-based synthesis (Area-Constrained Synthesis with Contamination avoidance, ACSC, presented in Section 8.4). Similar to RBSC, in ACSC we have also considered that the chip is partitioned into a number of equal areas, with a wash droplet assigned to each partition. However, compared to RBSC, the execution of an operation is constrained to an area on the chip, which is cleaned only the operation is completed.

The module library used for all experiments is shown in Table 2.1. Due to the large number of required wash droplets, we have considered that the method proposed in [45] will be used for dispensing droplets from the wash reservoirs. This method uses capacitance metering during the dispensing process, to produce up to 120 droplets per minute, while maintaining the reproducibility rate in a range of 10%. For sample and reagent liquids creating droplets with exact volume is important, as varying volumes can affect the integrity of the obtained result. However, as wash droplets are used just for cleaning the surface of the

[1] Values for the parameters: no_{part}=3, $max_{electrodes}$=50.

Area	Best		Average		Standard dev.	
	RBSC	ACSC	RBSC	ACSC	RBSC	ACSC
15 × 15	173.79	155.81	198.31	186.67	14.45	11.40
14 × 14	188.31	162.25	203.43	180.65	6.24	10.39
13 × 13	188.15	188.92	204.39	194.86	8.38	3.57

Table 8.3: Results for the colorimetric protein assay

chip, we assume that a 10% variation in volume is acceptable. Therefore, we consider that the dispensing of a wash droplet takes 0.5 s. For the rest of the liquids, the dispensing time has been set to 2 s, as shown in Table 2.1.

For all the experiments we have considered that at most four optical detectors can be integrated on the chip, together with one reservoir for sample liquid, two for buffer, two for reagent liquid and three for wash droplets. We have assumed that all operations except for the inputs containing buffer liquid will contaminate the surface of the biochip.

Table 8.3 presents the results obtained by using RBSC and ACSC for the synthesis of the colorimetric protein assay (see Section 2.4.3). Columns 2 and 3 in the table represent the best solution out of 50 runs (in terms of the application completion time $\delta_{\mathcal{G}}$) for RBSC and ACSC, respectively. The average and standard deviation over the 50 runs compared to the best application completion time are also reported in Table 8.3. The comparison is made for three progressively smaller areas. A time limit of 10 minutes was set for all experiments.

As we can see, when synthesizing applications in which contamination avoidance must be ensured, area-constrained routing leads to better results than transporting the droplets freely on the array. For example, in the case of the 14 × 14 array, constraining the movement of the operations to a group of electrodes leads to an improvement of 13.83% for the best schedule and 11.19% for the average length schedule obtained out of 50 runs. The main reason is the large number of electrodes contaminated during RBSC, when droplets are allowed to move on any route on the microfluidic array. This also leads to a high demand on the number of dispensed wash droplets, as contaminated electrodes must be cleaned as soon as possible in order not be block the execution of operations on the array. In contrast, in area-constrained synthesis a significantly smaller number of electrodes are contaminated while an operation is performed, and we can postpone cleaning these electrodes until the operation finishes executing.

In a second set of experiments we have compared RBSC with ACSC on three synthetic applications. The graphs are composed of 20, 40 and 60 operations

Operations	Area	Best		Average		Standard dev.	
		RBSC	ACSC	RBSC	ACSC	RBSC	ACSC
20	8 × 8	65.15	43.61	68.12	46.52	1.64	1.80
	7 × 8	71.09	46.90	76.62	50.55	2.14	2.14
	7 × 7	88.84	51.71	97.92	60.53	3.93	5.30
40	9 × 10	88.26	54.93	91.59	60.89	1.30	5.58
	9 × 9	91.28	57.01	97.78	65.42	1.80	5.06
	8 × 8	113.51	95.16	120.33	111.89	2.52	14.48
60	9 × 10	142.30	99.12	148.73	115.32	2.42	7.89
	9 × 9	151.80	110.27	158.20	127.54	2.75	10.87
	8 × 9	163.21	110	175.37	154.95	3.89	22.96

Table 8.4: Results for the synthetic benchmarks

and the results in Table 8.4 show the best and the average completion time, as well as the standard deviation obtained out of 50 runs for RBSC and ACSC, using a time limit of 10 minutes.

For each synthetic application we have considered three progressively smaller areas. The results shown in Table 8.4 confirm the conclusion from Table 8.3: when contamination is a concern, constraining the execution of operations to specific areas leads to better results in the average completion time. For example we have obtained an improvement of 38.18% in the average schedule length for the application with 20 operations, in the case of the 7 × 7 array.

Chapter 9

Conclusions and Future Directions

This chapter presents the conclusions of the thesis and discusses possible directions in which the proposed synthesis approaches can be extended.

9.1 Conclusions

In this thesis we have proposed several top-down synthesis techniques for digital microfluidic biochips. On such devices, biochemical operations such a mixing and dilution are performed on an array of electrodes, by routing the corresponding droplets. All previous work has assumed that an operation is constrained to a rectangular group of adjacent electrodes, forming a virtual device. All electrodes composing the module have been considered occupied throughout the execution of the operation, although the droplet uses one electrode at a time. However, an operation can be performed by transporting the corresponding droplet on any route on the microfluidic array.

The synthesis techniques proposed in this thesis are able to optimize the completion time of a biochemical application on a digital biochip, by considering the characteristic of dynamic reconfiguration of microfluidic operations. One of the

main conclusions of the thesis is that by relaxing the assumption that operation execution is constrained to fixed rectangular devices, significant improvements can be obtained, allowing us to use smaller biochips and thus reduce costs. The presented methods have been extensively evaluated using several real-life applications and synthetic benchmarks.

The conclusions are presented as follows.

- In Chapter 2 we have proposed an analytical method for determining the completion time of an operation on any given route. The method is based on the decomposition of a module library determined experimentally. The approach has been used throughout the thesis for determining completion times for non-rectangular devices (Section 6.2), devices with droplet-aware operation execution (Chapter 7) and operations executing on any route (Chapter 8).
- In Chapter 3 we have presented an Integer Linear Programming formulation for the problem of architectural-level synthesis and placement of DMBs, with fixed rectangular devices. Using two real-life examples, we have shown that our ILP-based approach can successfully synthesize small applications and find the optimal completion times, under given area constraints. Moreover, according to the results, considering placement at the same time with architectural-level synthesis leads to significant improvements in the completion time, compared to performing the two steps separately.
- In Chapter 5 we have presented a Tabu Search-based technique for the synthesis of digital microfluidic biochips. The proposed algorithm considers the unified architectural (allocation, binding and scheduling) and physical design (placement of operations on a microfluidic array). According to our evaluation, our approach can quickly obtain the optimal results for small-size applications and it can successfully synthesize larger applications, such as the colorimetric protein assay. We have compared our TS algorithm with the state of the art, the T-Tree topological representation [74] and we have shown that our approach can obtain better results, for the same design specifications.
- The Tabu Search algorithm was extended in Section 6.1 by considering that virtual devices can be moved during the execution of their operations. We have shown that by exploiting the dynamic reconfigurability of digital microfluidic biochips we can decrease the space fragmentation on the microfluidic array, improving the completion time of applications.
- In Section 6.2 we have further relaxed the assumption of virtual rectangular fixed devices by using modules of non-rectangular shape (e.g., "L"

shape). The effectiveness of this approach was evaluated using a real-life example as well as ten synthetic benchmarks. According to the obtained results, the space fragmentation on the microfluidic array is further reduced by allowing operations to execute on non-rectangular devices. Moreover, as the area decreases, considering dynamic reconfiguration becomes more important, and leads to significant improvements.

- In Chapter 7 we have proposed a module-based synthesis algorithm in which the positions of droplets inside devices are known at all times. This approach allows up to utilize better the chip area, since the accidental merging of droplets can be avoided without using segregation cells. We have evaluated our droplet-aware method using two real-life applications and three synthetic benchmarks and we have shown that controlling droplet movement during operation execution leads to a decrease in the application completion time, compared to the black-box approach.

- In Chapter 8 we have eliminated the concept of "virtual modules" and considered that operations can be performed on any route on the microfluidic array. The advantage of this routing-based approach is the increase in the parallelism, due to the fact that the same electrode can be used by several operations executing concurrently on the chip. According to the experiments, routing-based synthesis leads to significant improvements in terms of application completion time, compared to module-based synthesis. Moreover, routing-based synthesis is particularly important for more constrained synthesis problems. We have also shown that although the contamination problem can be successfully addressed for routing-based synthesis, constraining operation execution to a given area is to be preferred in this case.

9.2 Future Directions

The work presented in this thesis can be further extended to consider other challenges that remain to be tackled in the design of digital microfluidic biochips. Some of the directions in which the work can be extended are presented as follows.

9.2.1 Routing-Based Synthesis for Pin-Constrained DMBs

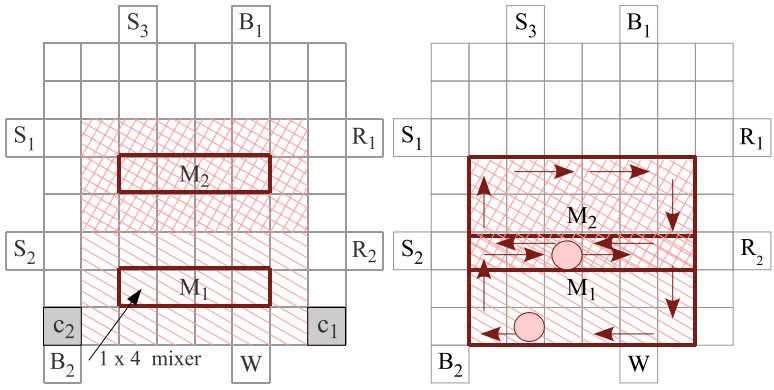
The algorithms presented in this thesis target direct addressing biochips, in which each electrode can be activated individually. However, as discussed in

Chapter 2, for larger biochips the increase in the number of required pins leads to high wiring complexity and increased fabrication costs. Several algorithms have been proposed so far for the synthesis problem of pin-constrained digital microfluidic biochips [23], [77], [28]. These algorithms consider that operations are executed inside rectangular virtual devices. As future work, the routing-based synthesis method proposed in Chapter 8 can be extended for pin-constrained biochips, by incorporating additional limitations regarding droplet movement. In the case of a pin-constrained biochip, several electrodes are activated using the same pin. Therefore, the decision regarding the direction in which a droplet is to be moved at time $t_{current}$ can no longer be made individually, as groups of droplets will be transported in the same direction with the activation of only one control pin. The modified algorithm must consider this additional constraint during the synthesis process.

9.2.2 Module-Based Synthesis with Overlapping Operations

The methods presented so far for module-based synthesis of DMBs have considered that devices placed on the microfluidic array cannot overlap.

We have shown in Chapter 7 that considering a droplet-aware operation execution approach leads to faster completion time of applications, due to a better utilization of the space on the microfluidic array. This can be further exploited by allowing devices on the microfluidic array to partially or even completely overlap. Let us consider the example in Figure 9.1b. The two mixers placed on



(a) Modules with segregation cells (b) Overlapping modules

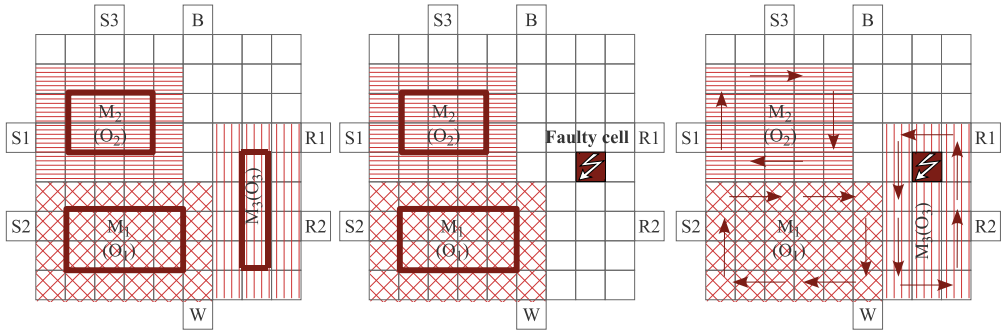
Figure 9.1: Module-based synthesis with overlapping devices

the array are sharing part of the electrodes composing the devices. However, we can avoid the accidental merging of the two droplets by controlling their movement at each step. This approach can be evaluated by extending the droplet-aware module-based synthesis proposed in Chapter 7 to consider the overlapping of devices during the operation execution.

9.2.3 Fault-Tolerant Module-Based Synthesis with Droplet-Aware Operation Execution

As discussed in Chapter 2, there are several types of faults that can affect the execution of a biochemical application on a digital biochip. For example, abnormal metal deposition during the fabrication process can lead to a failure in activating one or more electrodes on the microfluidic array, hindering the movement of droplets [69]. Therefore, testing methods such as the one proposed in [59] must be used in order to detect cells that have become faulty after the fabrication or during the operation of the biochip. Several fault-tolerant algorithms for module-based synthesis have been proposed so far [55], [70], [74]. These algorithms are based on partial reconfiguration, relocating modules, if needed, in order to avoid faulty cells. However, the disadvantage of reconfiguration during module-based synthesis is the fact that faulty cells can significantly increase the level of free space fragmentation on the microfluidic array.

Let us consider the example in Figure 9.2, where, at the current time-step t, three mixing operations O_1, O_2 and O_3 are scheduled to be executed on the microfluidic array. We assume that the operations are bound to the modules as follows: O_1 is bound to a 2×4 device, O_2 to a 2×3 device and O_3 to a 1×4



(a) Initial placement (b) Faulty cell (c) Droplet-aware synthesis

Figure 9.2: Fault tolerance during droplet-aware module-based synthesis

device. We ignore the positions of droplets inside the modules and wrap the devices in segregation cells, to avoid accidental mixing. If there are no defective electrodes on the array, the three mixers can be placed on the array at the same time, as shown in Figure 9.2a. However, let us assume that a faulty electrode has been detected, see Figure 9.2b. The fault-tolerant techniques proposed so far have considered faulty cells as obstacles during the placement of devices. Due to the fact that in black-box operation execution the position of droplets during execution is ignored, it has been considered that devices placed on the array can not overlap with faulty cells. This constraint guarantees that no operation will be executed on a defective cell. In case of Figure 9.2b this will lead to a delay in performing the mixing operation O_3, as the 1×4 module can not be placed on the array at time t such that it completely avoids the faulty cell. However, this delay can be avoided if we consider a module-based approach with droplet-aware operation execution. For example, in Figure 9.2c all the three mixing operations are executed concurrently. Even though the 1×4 module bound to operation O_3 contains the faulty cell, we can avoid using it by controlling the movement of the droplet inside the device at each time step.

Therefore, we believe that considering the positions of all droplets on the microfluidic array at each time step can improve the completion time of applications in the case of biochips with faulty electrodes. We consider the evaluation of this approach as a future work.

Bibliography

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design and Test of Computers*, 17(1):68–83, 2000.

[2] K. F. Bohringer. Towards optimal strategies for moving droplets in digital microfluidic systems. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1468–1474, 2004.

[3] K. Chakrabarty. Design automation and test solutions for digital microfluidic biochips. *IEEE Transactions on Circuits and Systems*, 57:4–17, 2010.

[4] K. Chakrabarty, R. B. Fair, and J. Zeng. Design tools for digital microfluidic biochips: Towards functional diversification and more than Moore. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(7):1001–1017, 2010.

[5] K. Chakrabarty and J.Zeng. *Digital Microfluidic Biochips: Synthesis, Testing, and Reconfiguration Techniques*. CRC Press, 2007.

[6] K. Chakrabarty and J. Zeng. Design automation for microfluidics-based biochips. *ACM Journal on Emerging Technologies in Computing Systems*, 1(3):186–223, 2005.

[7] M. Cho and D. Z. Pan. A high-performance droplet router for digital microfluidic biochips. In *Proceedings of the International Symposium on Physical Design*, pages 200–206, 2008.

[8] S. Dhar, S. Drezdon, and E. Maftei. Digital microfluidic biochip for malaria detection. Technical report, Duke University, 2008.

[9] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign*, pages 97–101, 1998.

[10] J. Ding, K. Chakrabarty, and R. B. Fair. Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20:1463–1468, 2001.

[11] R. B. Fair. Digital microfluidics: is a true lab-on-a-chip possible? *Microfluidics and Nanofluidics*, 3(3):245–281, 2007.

[12] R. B. Fair. Biochip engineering. University Lecture, 2008.

[13] R. B. Fair, A. Khlystov, T. D. Tailor, V. Ivanov, R. D. Evans, V. Srinivasan, V. K. Pamula, M. G. Pollack, P. B. Griffin, and J. Zhou. Chemical and biological applications of digital-microfluidic devices. *IEEE Design and Test of Computers*, 24(1):10–24, 2007.

[14] R. B. Fair, V. Srinivasan, P. Paik, V. K. Pamula, and M. G. Pollack. Electrowetting-based on-chip sample processing for integrated microfluidics. In *IEEE International Electron Devices Meeting*, pages 779–782, 2003.

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

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability–A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[17] Global Industry Analysts. `http://www.strategyr.com/`.

[18] F. Glover and M.Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[19] M. Gong and C.-J. Kim. Two-dimensional digital microfluidic system by multilayer printed circuit board. In *Proceedings of the Conference on Micro Electro Mechanical Systems*, pages 726–729, 2005.

[20] E. J. Griffith, S. Akella, and M. K. Goldberg. Performance characterization of a reconfigurable plannar array digital microfluidic system. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25:340–352, 2006.

[21] D. A. Hall, J. Ptacek, and M. Snyder. Protein microarray technology. *Mechanics of Ageing and Development*, 128:161–167, 2006.

[22] T.-W. Huang, C.-H. Lin, and T.-Y. Ho. A contamination aware droplet routing algorithm for digital microfluidic biochips. In *Proceedings of the International Conference on Computer-Aided Design*, pages 151–156, 2009.

[23] W. Hwang, F. Su, and K. Chakrabarty. Automated design of pin-constrained digital microfluidic arrays for lab-on-a-chip applications. In *Proceedings of the Design Automation Conference*, pages 925–930, 2006.

[24] International Technology Roadmap for Semiconductors. `http://www.itrs.net/Links/2007ITRS/Home2007.htm`.

[25] H. G. Kerkhoff. Testing microelectronic biofluidic systems. *IEEE Design and Test of Computers*, 24(1):72–82, 2007.

[26] M. U. Kopp, A. J. de Mello, and A. Manz. Chemical amplification: Continuous-flow pcr on a chip. *Science*, 280(5366):1046–1048, 1998.

[27] M. F. Kramer and D. M. Coen. Enzymatic amplification of DNA by PCR: Standard procedures and optimization. *Current Protocols in Molecular Biology*, pages 15.1.1–15.1.14, 2001.

[28] C. C. Y. Lin and Y.-W. Chang. Cross-contamination aware design methodology for pin-constrained digital microfluidic biochips. In *Proceedings of the Design Automation Conference*, pages 641–646, 2010.

[29] C. C.-Y. Lin and Y.-W. Chang. ILP-based pin-count aware design methodology for microfluidic biochips. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(9):1315–1327, 2010.

[30] E. Maftei, P. Paul, and J. Madsen. Tabu search-based synthesis of dynamically reconfigurable digital microfluidic biochips. In *Proceedings of the Compilers, Architecture, and Synthesis for Embedded Systems Conference*, pages 195–203, 2009.

[31] E. Maftei, P. Paul, and J. Madsen. Routing-based synthesis of digital microfluidic biochips. In *Proceedings of the Compilers, Architecture, and Synthesis for Embedded Systems Conference*, pages 41–49, 2010.

[32] E. Maftei, P. Paul, and J. Madsen. Tabu search-based synthesis of digital microfluidic biochips with dyanmically reconfigurable non-rectangular devices. *Journal of Design Automation for Embedded Systems*, 14:287–308, 2010.

[33] E. Maftei, P. Paul, J. Madsen, and T. Stidsen. Placement-aware architectural synthesis of digital microfluidic biochips using ILP. In *Proceedings of the International Conference on Very Large Scale Integration of System on Chip*, pages 425–430, 2008.

[34] E. Maftei, P. Pop, and J. Madsen. Module-based synthesis of digital microfluidic biochips with droplet-aware operation execution. *Journal on Emerging Technologies in Computing Systems*, Under review.

[35] J. Melin and S. R. Quake. Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annual Review of Biophysics and Biomolecular Structure*, 36:213–231, 2007.

[36] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Science, 1994.

[37] E. Miller and A. R. Wheeler. Digital bioanalysis. *Analytical and Bioanalytical Chemistry*, 393(2):419–426, 2009.

[38] H. Moon, A. R. Wheeler, R. L. Garrell, J. A. Loo, and C.-J. Kim. An integrated digital microfluidic chip for multiplexed proteomic sample preparation and analysis by MALDI-MS. *Lab on a chip*, 6(9):1213–1219, 2006.

[39] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[40] P. Paik, V. K. Pamula, and R. B. Fair. Rapid droplet mixers for digital microfluidic systems. *Lab on a Chip*, 3:253–259, 2003.

[41] V. K. Pamula, V. Srinivasan, H. Chakrapani, R. B. Fair, and E. J. Toone. A droplet-based lab-on-a-chip for colorimetric detection of nitroaromatic explosives. *Proceedings of the Micro Electro Mechanical Systems Conference*, pages 722–725, 2005.

[42] M. G. Pollack, P. Y. Paik, A. D. Shenderov, V. K. Pamula, F. S. Dietrich, and R. B. Fair. Investigation of electrowetting-based microfluidics for real-time pcr applications. *μTAS*, pages 619–622, 2003.

[43] M. G. Pollack, A. D. Shenderov, and R. B. Fair. Electrowetting-based actuation of droplets for integrated microfluidics. *Lab on Chip*, 2:96–101, 2002.

[44] Professor Quake's Group at Stanford University. `http://thebigone.stanford.edu/`.

[45] H. Ren and R. B. Fair. Micro/nano liter droplet formation and dispensing by capacitance metering and electrowetting actuation. In *Proceedings of the IEEE-NANO*, pages 369–372, 2002.

[46] H. Ren, V. Srinivasan, and R. B. Fair. Design and testing of an interpolating mixing architecture for electrowetting-based droplet-on-chip chemical dilution. In *Proceedings of the International Conference on Transducers, Solid-State Sensors, Actuators and Microsystems*, pages 619–622, 2003.

[47] A.J. Ricketts, K. Irick, N. Vijaykrishnan, and M.J. Irwin. Priority scheduling in digital microfluidics-based biochips. In *Proceedings of the Design, Automation and Test in Europe*, volume 1, pages 1–6, 2006.

[48] C. Hashi S.-K. Fan and C. J. Kim. Manipulation of multiple droplets on N × M grid by cross-reference ewod driving scheme and pressure-contact packaging. In *Proceedings of the International Conference on MEMS*, pages 694–697, 2003.

[49] Oliver Sinnen. *Task Scheduling for Parallel Systems.* Wiley, 2007.

[50] V. Srinivasan, V. K. Pamula, and R. B. Fair. Droplet-based microfluidic lab-on-a-chip for glucose detection. *Analytica Chimica Acta*, 507:145–150, 2004.

[51] V. Srinivasan, V. K. Pamula, and R. B. Fair. An integrated digital microfluidic lab-on-a-chip for clinical diagnostics on human physiological fluids. *Lab Chip*, 4:310–315, 2004.

[52] R. B. Stoughton. Applications of DNA microarrays in biology. *Annual Review of Biochemistry*, 74:53–82, 2005.

[53] F. Su and K. Chakrabarty. Architectural-level synthesis of digital microfluidics-based biochips. In *Proceedings of the International Conference on Computer Aided Design*, pages 223–228, 2004.

[54] F. Su and K. Chakrabarty. Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips. In *Proceedings of the Design Automation Conference*, pages 825–830, 2005.

[55] F. Su and K. Chakrabarty. Module placement for fault-tolerant microfluidics-based biochips. *ACM Transactions on Design Automation of Electronic Systems*, 11(3):682–710, 2006.

[56] F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. *Journal on Emerging Technologies in Computing Systems*, 3, 2008.

[57] F. Su, W. Hwang, and K. Chakrabarty. Droplet routing in the synthesis of digital microfluidic biochips. In *Proceedings of the Design, Automation and Test in Europe*, volume 1, pages 73–78, 2006.

[58] F. Su, W. Hwang, A. Mukherjee, and K. Chakrabarty. Defect-oriented testing and diagnosis of digital microfluidics-based biochips. In *Proceedings of the International Test Conference*, pages 487–496, 2005.

[59] F. Su, S. Ozev, and K. Chakrabarty. Testing of droplet-based microelectrofluidic systems. In *Proceedings of the International Test Conference*, pages 1192–1200, 2003.

[60] F. Su, S. Ozev, and K. Chakrabarty. Concurrent testing of droplet-based microfluidic systems for multiplexed biomedical systems. In *Proceedings of the International Test Conference*, pages 883–892, 2004.

[61] F. Su, S. Ozev, and K. Chakrabarty. Ensuring the operational health of droplet-based microelectrofluidic biosensor systems. *IEEE Journal on Sensors*, 5:763–773, 2005.

[62] P. Tabeling. *Introduction to microfluidics*. Oxford University Press, 2006.

[63] T. Thorsen, S. Maerkl, and S. Quake. Microfluidic largescale integration. *Science*, 298:580–584, 2002.

[64] D. Ullman. NP-complete scheduling problems. *Journal of Computing System Science*, 10:384–393, 1975.

[65] W. Wang, Z. X. Li, Y. J. Yang, and Z. Y. Guo. Droplet based micro oscillating flow-through pcr chip. *Proceedings of the International Conference on Micro Electro Mechanical Systems Conference*, pages 280–283, 2004.

[66] Z. Xiao and E. F. Y. Young. Crossrouter: A droplet router for cross-referencing digital microfluidic biochips. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 269–274, 2010.

[67] T. Xu and K. Chakrabarty. Droplet-trace-based array partitioning and a pin assignment algorithm for the automated design of digital microfluidic biochips. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 112–117, 2006.

[68] T. Xu and K. Chakrabarty. A cross-referencing-based droplet manipulation method for high-throughput and pin-constrained digital microfluidic arrays. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 552–557, 2007.

[69] T. Xu and K. Chakrabarty. Functional testing of digital microfluidic biochips. In *Proceedings of the International Test Conference*, pages 1–10, 2007.

[70] T. Xu and K. Chakrabarty. Integrated droplet routing and defect tolerance in the synthesis of digital microfluidic biochips. In *Proceedings of the Design Automation Conference*, pages 948–953, 2007.

[71] T. Xu and K. Chakrabarty. Parallel scan-like testing and fault diagnosis techniques for digital microfluidic biochips. In *Proceedings of the European Test Symposium*, pages 63 – 68, 2007.

[72] T. Xu and K. Chakrabarty. Broadcast electrode-addressing for pin-constrained multi-functional digital microfluidic biochips. In *Proceedings of the Design Automation Conference*, pages 173 – 178, 2008.

[73] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang. Bioroute: A network-flow-based routing algorithm for the synthesis of digital microfluidic biochips. In *Proceedings of the International Conference on Computer-Aided Design*, pages 752–757, 2007.

[74] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang. Placement of defect-tolerant digital microfluidic biochips using the T-tree formulation. *ACM Journal on Emerging Technologies in Computing Systems*, 3(3), 2007.

[75] Y. Zhao and K. Chakrabarty. Cross-contamination avoidance for droplet routing in digital microfluidic biochips. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1290–1295, 2009.

[76] Y. Zhao and K. Chakrabarty. Synchronization of washing operations with droplet routing for cross-contamination avoidance in digital microfluidic biochips. In *Proceedings of the Design Automation Conference*, pages 641–646, 2010.

[77] Y. Zhao, R. Sturmer, K. Chakrabarty, and V. K. Pamula. Synchronization of concurrently-implemented fluidic operations in pin-constrained digital microfluidic biochips. In *International Conference on VLSI Design*, pages 69–74, 2010.