UNIVERSITY OF CALIFORNIA
RIVERSIDE

Placement, Routing, and Post-Processing of Microfluidic Device Flow-Layers

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Brian Russell Richard Crites

September 2018

Dissertation Committee:

Dr. Philip Brisk, Chairperson
Dr. William Grover
Dr. Jiasi Chen
Dr. Tamar Shinar

The Dissertation of Brian Russell Richard Crites is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to thank my adviser, Professor Philip Brisk, without whom this project would not exist, and Professor William Grover, who was essential in moving this project from theory into real-world application. I would also like to thank my other committee members: Professors Jiasi Chen, Tamar Shinar, and Eamonn Keogh, who asked new and insightful questions no matter how many times they heard the material. I would like to thank Dr. Jeffrey McDaniel, who started this project and my involvement in it, and Heran Bhakta, who had the patience to explain all the biology and physics we were getting wrong.

I would like to thank my other colleagues over the years: Chris Curtis, Tyson Loveless, Kenneth O'Neil, Jason Ott, Aditya Tammewar, and Zachary Zimmerman; who always made the time to "come over and take a look at this for a minute," no matter how many minutes it took. I would also like to thank my collaborators at Boston University Joshua Lippai and Radhakrishna Sanka, under the direction of Professor Douglas Densmore, who were instrumental partners in the development of the ParchMint benchmarking suite and are generally amazing collaborators.

I would like to thank UC Riverside, which took me as a graduate student even after all the trouble I was in undergrad. I would also like to thank all the undergraduate students I have had the privileged of working with there, who are too many to list here. Without their efforts many of these projects would not have been possible and none would have been as good.

Finally, I would like to thank my family, friends, and loved ones. Without their support I would never have gotten this far.

A number of previous publications were used in the preparation of this dissertation, and parts of them are reprinted with the following permissions.

To my family, who let me push every button, turn every dial, and flip every switch.

ABSTRACT OF THE DISSERTATION

Placement, Routing, and Post-Processing of Microfluidic Device Flow-Layers

by

Brian Russell Richard Crites

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2018
Dr. Philip Brisk, Chairperson

Continuous flow based microfluidic devices have made great strides in fields like rapid DNA sequencing and ex-vivo tissue samples, so-called organ-on-a-chip devices, for biological testing. While a large number of new components and biological processes have been developed, tools to help design these devices have not followed suit. As the field grows and the devices being designed become more complex, they will quickly become too difficult for a single person or small group to develop without computer assistance. This necessitates the development of tools and algorithms that can accelerate or automate parts of the design process. This dissertation presents and evaluates a collection of algorithms that form the crux of a larger software platform for microfluidic design. Algorithms are presented here for automated flow layer design and post-processing for area reduction and to automate the process of high-throughput conversions. It concludes by introducing a suite of benchmarks and design metrics to facilitate unbiased comparisons between the microfluidic design automation algorithms introduced here, and future work in the space to be performed by others.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Microfluidic Devices & Principles

Devices based on continuous fluid flow microfluidics are used for a wide variety of biochemical applications including high-throughput screening [23], protein crystallization [20], long-term single cell culturing and monitoring [5], single-cell mRNA isolation and DNA synthesis [36], single-cell tracking and imaging [14], solid-phase capture immunoassays [30], and interrogation of protein-DNA interactions [67], among many others. Through automation and miniaturization, microfluidic devices offer the benefits of higher throughput, lower sample/reagent usage, and reduced likelihood of human error compared to traditional benchtop chemistry methods that they often replace. Additionally, their scale and automation can allow them to perform more complex and sophisticated processes than those that can typically be performed by with traditional methods. One long-term objective of the scientific movement to design and commercialize microfluidic devices is to create low-cost point-of-

care testing devices that can positively impact global health, especially in the developing world [70].

Generally, microfluidic devices are characterized by having fluid flows that fall within the laminar flow regime due to their typically low Reynolds numbers Re.

$$Re = \frac{\rho u L}{\mu}$$

Here, ρ is the density of the fluid in the device, u is the velocity of the fluid, L is the fluid channel width, and μ which is the dynamic viscosity of the fluid [53]. Because the fluids used within an experiment are more or less fixed (with small exceptions for cases where modifiers like surfactants can be used) the ρ and μ variables are relatively fixed. Since the fluids of interest are typically biological in nature, the ρ and μ values are typically close to that of water. This means that microfluidic devices operate in the laminar regime because of their relatively small channel widths L and low fluid velocities u. As long as the Reynold number $Re < 1$, the flows within the device will be in the laminar regime and the fluid, and any material that the fluid is carrying, will travel in relatively straight parallel lines.

Continuous flow microfluidic devices can be broken into two major categories. The first and most popular form, especially when considering microfluidic products on the market, are *passive devices*. Passive microfluidic devices are, in their most basic form, systems of channels that may be etched or carved into a rigid substrate [29, 12, 49] or imprinted in a flexible polymer [68] and mounted on a rigid substrate. Through the application of flow pressure, either head or provided through an external pump, fluids move through these

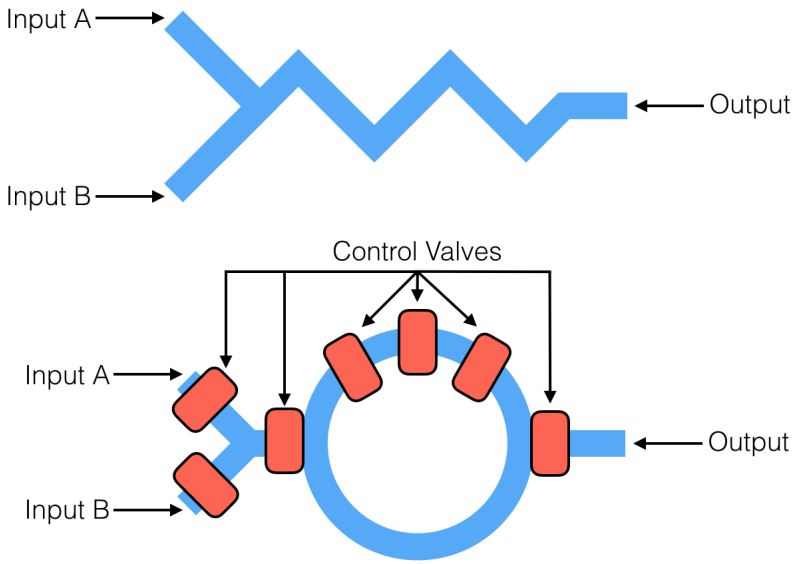


Figure 1.1: A passive serpentine mixer (top), the shape of which induces turbulence causing the two input flows to mix and a rotary mixer (bottom) which uses the peristaltic pumping to mix.

channels in order to accomplish some biological or chemical process of interest. The primary means of accomplishing these processes is through the use of specific geometries in the channels and chambers that make up the device.

Using knowledge of the medium of interest that will be used in the device and the fluids used for transporting those mediums, the geometry of each component is designed to ensure that the component can perform a particular action. For example, the serpentine channel mixer shown in Figure 1.1 uses diffusion over a longer distance to create fluid mixing. While some devices are designed to perform a single basic task, there are increasing numbers of devices that integrate a number of different processes to perform increasingly complex and sophisticated biological and chemical procedures.

The second major category of devices are *active devices*. These types of devices are capable of utilizing the same geometric designs to perform a specific process, but add an additional layer whose primary purpose it to externally control the flow of fluids through the microfluidic channels. They key enabling technology of active devices is the microvalve, the fluidic analogue to the transistor. Microvalves allow external pressure sources to actuate on-device valves, which can be used to start and stop the flow of fluid to create timing as well as perform complex functions [60].

1.2 The Microvalve

A typical active device is comprised of two or more layers of ridged substrate separated by a flexible membrane. In a typical two- or three-layer device, there is one "flow layer" which transports biological medium and other transport fluid(s), and one or two control layers which use pressure to start and stop the flow of fluids in the flow layer through integrated microvalves. A microvalve (Figure 1.2) is formed at any point where a control channel on one layer crosses a flow channel on another layer and that channel is large enough and/or the pressure high enough to lead to a deformation in the flexible membrane. This deformation causes the flexible membrane to block the flow of medium through the flow layer. Figure 1.2 illustrates the two major technologies for microvalves.

The microvalve technology illustrated at the top of Figure 1.2 is one version of the *multi-layer soft lithography valve* [60]. Here, two layers of rigid substrate are separated by a flexible membrane. The flow layer containing channels for fluid movement is on bottom, while the control layer above provides actuation; however, the layer ordering can be

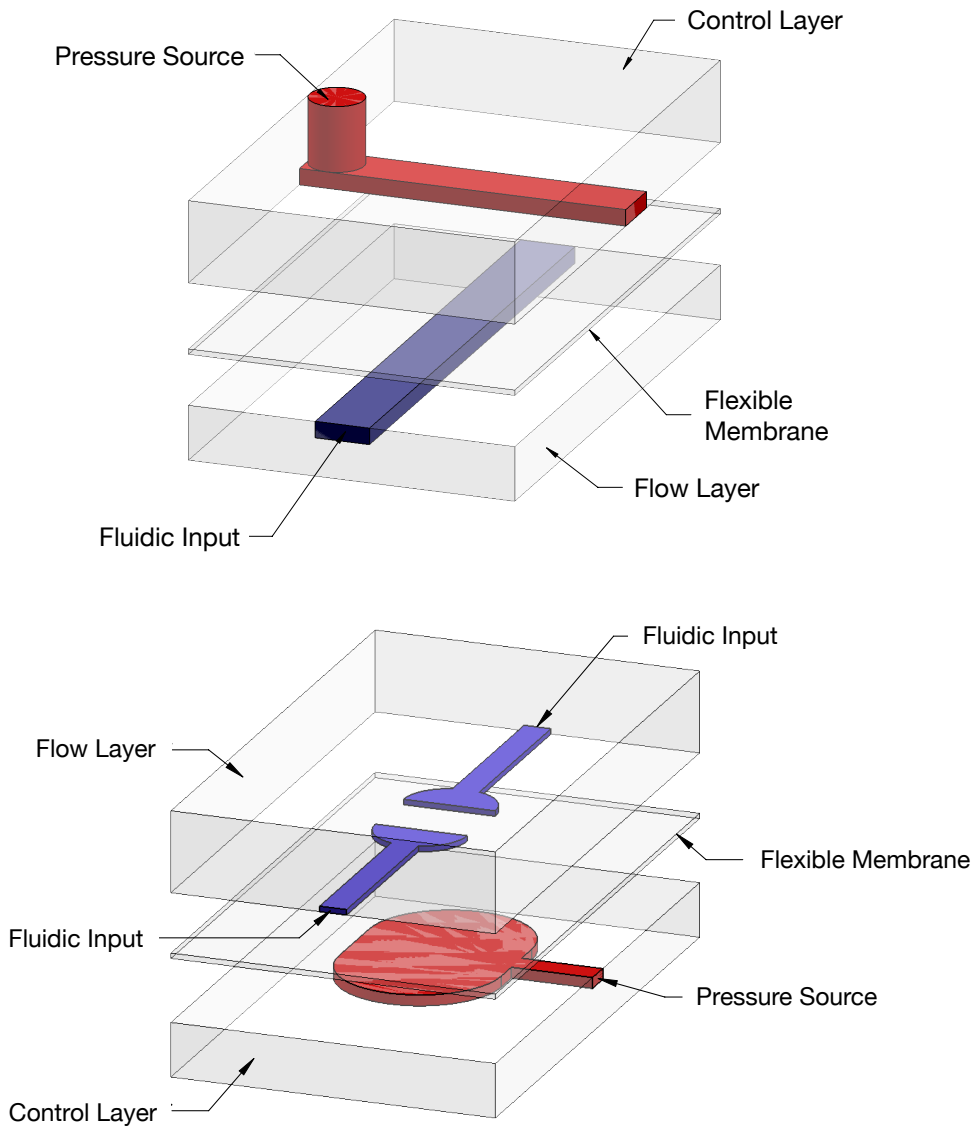


Figure 1.2: Microvalves in the multi-layer soft lithography (top) and monolithic membrane (bottom) technologies

reversed. In a more typically configuration, this devices is made of two layers of a flexible polymer substrate, usually polydimethylsiloxane (PDMS), which are mounted on top of a rigid substrate such as a glass slide. This configuration allows for the flow layer to be deformed by the control layer, making it easier to actuate the valve. By default, these types

of microvalves are open. When the control layer and flow layer meet, if there is a large enough area of intersection and/or a high enough pressure the flexible membrane (or flow layer) is forced to deflect blocking the movement of fluid.

The microvalve technology illustrated at the bottom of Figure 1.2 is *monolithic membrane valve* [19]. This valve is constructed in a similar fashion to the multi-layer soft lithography valve with flow and control layers constructed using a ridged substrate, separated by a flexible membrane. The primary difference in this valve is in its operation. The monolithic membrane valve is by default closed, with the flexible membrane resting on the gap etched between the two pieces of the flow channel. When a vacuum, rather than pressure, is applied to the control channel the flexible membrane deflects away from the flow layer towards the control layer, opening a space for fluid to flow through the now complete channel. While there are still minimum size and pressure requirements necessary in order to cause the valve to actuate, the amount of pressure (or in this case vacuum) is typically far less than what is necessary in a multi-layer soft lithography valve. Additionally, if the multi-layer soft lithography valve is fabricated in the more typical fashion from two layers of PDMS, then monolithic membrane valves reduce the amount of contact that fluids have with the PDMS. There are a number of issues that can occur when biological and chemical fluids interact with PDMS, such as swelling or disintegrating of the PDMS in solvents or leaching of monomers into a reactions ruining it.

Microvalves are of great interest to microfluidic designers because they give two primary benefits: (1) they allow for precise control of timing withing a microfluidic device, which makes biological or chemical processes containing multiple steps much easier and (2)

they allow for direct control of fluids within the device, either for intermittent movement or for switching between different available paths.

1.3 Microfluidic Very-large-scale Integration

This interest in integrating an ever increasing number of microvalves into a device to create ever more complex processes has lead to the development of microfluidic very-large-scale integration (mVLSI). The concept of mVLSI is directly related to the concept of very-large-scale integration (VLSI) for microprocessors, which is the every increasing need to fit more and more transistors onto a single device in order to drive forward compute power and/or reduce device size. Modern research devices can integrate hundreds or thousands of externally controllable microvalves [60, 19, 43, 3].

Similar to how several transistors can be assembled to implement arbitrary logic functions, sets of microvalves can be organized to form larger more useful components. Figure 1.1 (bottom) shows a rotary mixer which uses a number of valves to push two different fluids into the top and bottom segments of the ring channel, peristaltically pumping the ring while fluidically separating it from the rest of the device, and then flowing the mixed fluid out of the ring. Combinations of valves and channels can create complex components such as pressurized latches, multiplexers, demultiplexers, memories, and logic gates [43]. These components can then be assembled and connected to form fully integrated devices. While these active devices show great promise, the vast majority of the devices currently being fabricated in industry are passive devices or active devices with very few microvalves.

There are a number of reasons for including the increased costs associated with multi-layer devices. One is the need for additional equipment to run the device which may not be available, especially for field deployed devices. Another is the reliability of valves, especially over time and with different fluids. However, one of the primary causes is the sheer amount of complexity that is required in order to create a device that takes advantage of thousands of microvalves. This has lead to an increasing interest in creating software capable of automating different aspects of the microfluidic design process similar to the software that was developed to solve the challenges of VLSI design.

1.4 Limitations of Current Design Methods

At present, both layers of mVLSI devices are manually designed using software such as SolidWorks or AutoCAD. Manual design is tedious, error-prone, and unlikely to scale as integration densities increase. Using the semiconductor industry as a metaphor, the current design paradigm is stuck in the early 1970s, before the Conway-Mead revolution led to integrated semiconductor VLSI technology and computer-aided design tools, which are now industry standard [42]. There is a fundamental limit to the microfluidic design complexity that can be achieved by manual layout, and the most advanced devices that have been designed and fabricated, to date, are readily approaching that limit.

mVLSI physical design is challenging because components are heterogeneous in terms of size and dimensions and can be placed at any location and with any orientation on the chip. This is distinct from semiconductor VLSI which follows standard design rules [42] and where standard cells have a uniform height and are placed in rows. This means that

while we use VLSI as an analog for mVLSI, established physical design techniques cannot easily be adapted for microfluidic technology.

As the sophistication of microfluidic devices increases, new tools and algorithms are necessary in order for microfluidic designers to be able to cope with the complexity. In the following chapters we propose a number of different algorithms to automate or accelerate the design process of the microfluidic flow layer. Where applicable, we discuss the effects that these algorithms have on the control layer.

Chapter 2

Planar Embedding Based Placement & Routing

2.1 Planarity in Device Designs

In very-large-scale integration (VLSI), many different layers of printed circuit board (PCB) containing electrical components and connections are stacked on top of each other to form a single device. When a component from one layer needs to connect with a component on another, a through layer *via* is used in order to allow that connection to cross the intermediate layers. While microfluidic devices can contain through layer vias for both the flow and control layers, allowing fluids to move freely between layers, these vias are undesirable. The reasons are similar to those for the inclusion of microvalves; the cost of the device increases as more layers are added and the reliability is reduced because of bursting or leakage where vias connect two layers.

Because of this one of the challenges of manual design today is to produce a planar layout for the flow layer which does not compromise the desired functionality of a chip. In the context of design automation, this means that only chips having planar architectures can be fabricated. Two primary approaches to microfluidic very-large-scale integration (mVLSI) placement were proposed before this work: Incremental Cluster Expansion (ICE) [56] and Simulated Annealing (SA) [45, 41]. ICE selects sets of components that should be placed near one another to simplify the mVLSI chip architecture, while SA uses randomization and iterative improvement to reduce the number of intersections. These methods attempt to reduce the total area necessary to place all components as well as trying to find a good starting point for routing. The mVLSI routing algorithms proposed before this work attempted to minimize the total fluid channel length, but did not consider the issue of planarity, or legality of chip for fabrication. Proposed techniques include variants of Hadlock's Algorithm for grid routing [45, 41], Dijkstra's Algorithm [56], and Steiner Tree construction [35]. Because these algorithms were designed to reduce total channel length, they allowed channels to cross in order to meet that goal.

These methods are not focused on directly finding a planar design, but have instead corrected any intersections created in the flow layer by the placement and routing process through the introduction of additional switches as shown in Figure 2.1 [41]. This switch is made up of four separate microvalves (Figure 2.2), with each opposing pair of microvalves connected a control line. While this method is theoretically sound, there are a number of issues with it in practice. The inclusion of a switch necessitates a control layer in order to actuate the microvalves, meaning these methods cannot be used to generate a passive

device. Additionally, the number of input and output (I/O) ports which can be fabricated onto a device are limited in practice, effectively limiting the number of switches that can be inserted. Finally, because each switch introduces two additional control lines each inserted switch increases the difficulty of finding a valid set of routes for the control layer.

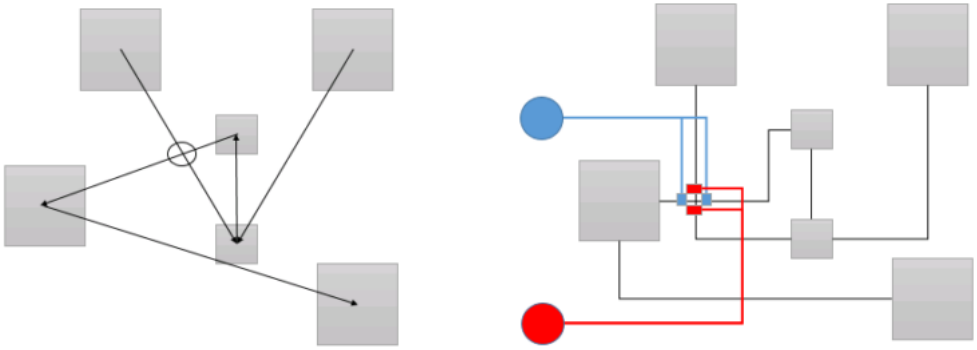
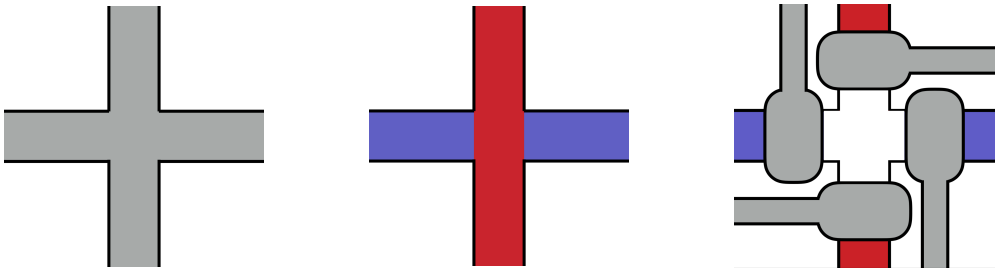


Figure 2.1: When the routes are non-planar, 4-way switches must be added to compensate, adding to control system complexity and fabrication cost



(a) (b) (c)

Figure 2.2: (a) two channels that intersect during routing, if left uncorrected (b) lead to the two fluids intersecting when the device is running. This is corrected by (c) inserting switches to separate the flows of fluids in the channels.

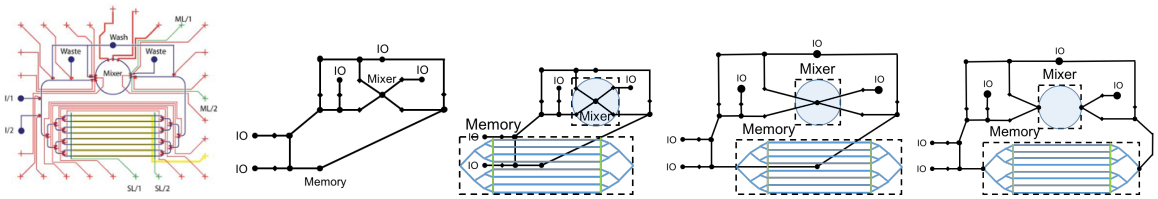
Because of these issues, methods to create a planar flow layer, one that introduces no intersections, are necessary for the automation design of passive devices and desirable for active devices. Here we take inspiration from planar embedding algorithms in graph

theory, which are capable of taking an abstract netlist of nodes and edges between those nodes laying them out in a planar fashion such that no two edges intersect and nodes only intersect edges where designated. However, these algorithms are insufficient for direct usage as a placement step. Planar embeddings treat vertices as points, whereas in mVLSI technology, each component has two dimensions with area.

To address this limitation, we have augmented an existing planar embedding algorithm with a number of post-placement processing steps that accounts for component sizes. We also introduce a planar fluid channel routing algorithm that performs port assignment as a co-optimization while disallowing intersections. In this chapter, we evaluate these algorithms on a set of planar benchmarks taken from mVLSI chips that have been designed and laid out by hand, along with a small set of synthetic planar designs. Our experiments demonstrate that our algorithm can obtain legal planar embeddings that adhere to the design rules laid out by the Stanford Microfluidics Foundry [2]. In contrast, existing algorithms that have been proposed for mVLSI placement and routing do not obtain planar layouts, and therefore yield solutions that violate foundry design rules and cannot be fabricated.

2.1.1 Motivating Example

Figure 2.3a shows a microfluidic device that was designed manually and published in 2006 [61]. The approach advocated here is to start with a language-based specification of the device, e.g., using microfluidic hardware design language (MHDL) [39], which is then compiled into a netlist, represented by a graph. To produce the layout, the first step is to compute a



(a) (b) (c) (d) (e)

Figure 2.3: (a) A programmable mVLSI device laid out manually [61]; (b) planar embedding of a mVLSI netlist representation of the device where no components or routes overlap or intersect, making it planar; (c) component expansion after planar embedding yields an illegal layout, where several microvalves and I/O ports overlap with the expanded memory and mixer components; (d) shifting the positions of expanded components yields a legal layout, which is larger but similar to the original planar embedding; and (e) the legal layout after routing and port recovery, with valid port-to-port connections.

planar embedding of the graph as shown in Figure 2.3b, which yields a planar layout with single points representing the components. For this particular netlist, two components, a rotary mixer and a memory, are considerably larger than I/Os and microvalves. Given this layout, expanding the vertices to represent the actual dimensions of the components creates an illegal layout, as shown in Figure 2.3c, due to multiple components and fluid channels overlapping. To legalize this placement, many of the components must be moved to new positions to accommodate the fully expanded mixer and memory, as shown in Figure 2.3d, while trying to maintain as much of the original planar embedding as possible. The final step is to identify the locations of I/O ports on the perimeters of the expanded components, and to route fluid channels to the I/O ports, as opposed to the centroids of the components, as shown, in Figure 2.3e. This process produces a legal, although not necessarily optimal, flow layer design.

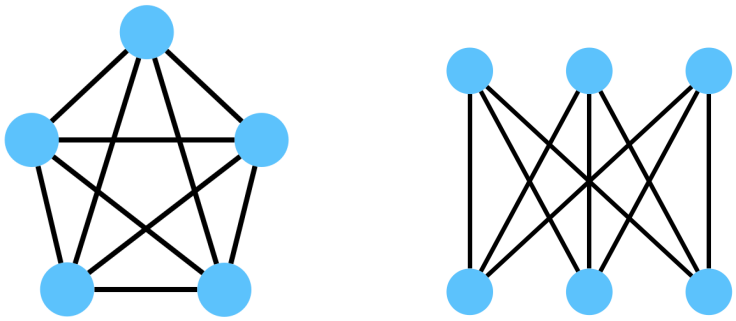
2.2 Preliminaries

2.2.1 Graph Abstraction

The architecture of an mVLSI device can be viewed as a netlist of components and the connections between them. Each component $c_i \in C$ in the netlist is defined as a tuple $c_i = (T_i, P_i, x_i, y_i, w_i, h_i)$, where (x_i, y_i) is the coordinate for the upper left corner, w_i and h_i are the width and height respectively, $t_j \in T$ corresponds to the component c_j being connected to c_i, and P_i is the set of ports on the component. The port locations $p_j \in P_i$ are the only locations on the component to which channels can connect. Non-rectangular components, such as circular mixers, are approximated by rectangular components that represent their bounding boxes.We use the (x_i, y_i) coordinate to represent the component as a single point in the graph abstraction, G, during the placement phase of the physical synthesis.

The netlist is represented by a graph $G = (V, E)$, where vertex $v_i \in V$ represents component $c_i \in C$; and edge $e_i = (v_i, v_j) \in E$ represents a fluidic channel that connects v_i and v_j. The primary difference between vertices and components is that vertices are points while components are rectangles. Our approach to placement starts with vertices and replaces them with components while preserving the desired properties of planar layout.

2.2.2 Planar Graphs



(a) (b)

Figure 2.4: The Kuratowski subgraphs (a) K_5 and (b) $K_{3,3}$.

A graph G is planar if it can be *embedded* in the plane, i.e., if it can be drawn on the plane in such a manner that edges only cross at their endpoints. In the context of mVLSI, this means that the graph can be placed and routed such that fluid routing channels intersect only at components (represented, for now, as points). Every planar graph also admits a *straight line planar embedding* in which the planar graph property is preserved and all edges can be drawn as straight line segments.

Algorithmic planarity testing is typically based on an alternate, but equivalent definition of planarity: graph G is planar if and only if it does not contain the specific graphs K_5 or $K_{3,3}$, Figure 2.4, as minors, where a minor is a graph H that can be obtained from G by deleting vertices and/or deleting or contracting edges [31].

2.2.3 Implications for mVLSI Technology

A legal placement and routing solution for the flow layer of an mVLSI chip is essentially a planar embedding that treats vertices as components with dimensions, rather than points. In this context, a legal planar embedding means that components do not overlap one another, routed fluid channels do not intersect components or other fluid channels. Our approach is to take a planar graph embedding and convert it into a planar mVLSI layout.

2.3 Planar Placement

Methods that utilize a planar embedding as a starting point for their placement step have generally come to be referred to as "Planar Placement" methods. While this term can refer to a planar embedding paired with any of the expansion methods presented here, when we reference the term "Planar Placement" later in this dissertation we are referring specifically to the Baseline Expansion method described in Section 2.4.1 unless otherwise stated.

2.3.1 Straight Line Planar Embedding

The process starts with a graph $G = (V, E)$ representing the netlist of components and their connections; vertices do not yet have dimensions or area. The first step is to make G fully connected, and check for planarity using the Boyer-Myrvold method [6]. If G is planar, then it is transformed to be biconnected and maximally planar. The vertices $v_i \in V$ are then ordered canonically and the Chrobak-Payne straight line embedding algorithm [8]

Require: $G := (V, E)$ an undirected graph
Ensure: $G := (V, E)$ with each $v_i \in V$ placed
1: $G := make\_connected(G)$
2: **if** $!boyer\_myrvold\_planarity\_test(G)$ **then**
3: $exit()$
4: **end if**
5: $G := make\_biconnected\_planar(G)$
6: $G := make\_maximal\_planar(G)$
7: $X := planar\_canonical\_ordering(G)$
8: $G := chrobak\_payne\_straight\_line(G, X)$

Figure 2.5: Pseudocode for the Chrobak-Payne straight line embedding from the Boost library. The function calls shown here are Boost library calls.

is invoked to obtain a straight line planar embedding. Our implementation of these steps uses the Boost Library; Figure 2.5 provides a high-level overview.

2.3.2 Component Expansion

The straight line embedding does not account for the dimensions of components. To create a valid mVLSI embedding, we must apply a component expansion technique in order to expand components and remove any overlap between them. Here we explore a number of different techniques for component expansion which are compared in Section 2.8.3.

2.4 Component Expansion Methods

2.4.1 Baseline Expansion

The Baseline Expansion (BaseEx) method makes two passes over the set of components to perform the expansion. The first pass sorts the components $c_i \in C$ by their x_i coordinate in ascending order, and expands each component by its width w_i. All subsequent components $c_j \in C$, where $j > i$, are shifted in the positive x direction by w_i; $x_j = x_j + w_i$. The

second pass of the expansion applies the same steps along on the y-axis, while expanding and shifting components based on their heights, rather than their widths. This method of expansion is relatively naive, however by moving every other component by the full width and height of the component being expanded, we guarantee that there can be no overlap between the components as shown in Figure 2.6.

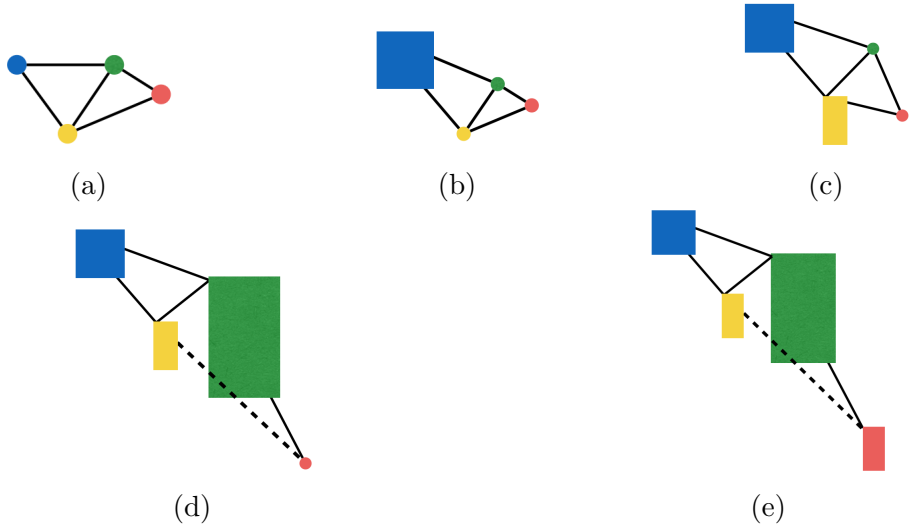


Figure 2.6: (a) The original graph, (b-e) expands the components one at a time to their full size. The dotted line represents the straight line connection that has been invalidated because of the expansion.

In practice, we have observed that component expansion rarely preserves the straight line planar embedding that was computed previously. Figure 2.6 shows how the component expansion invalidates the straight line planar embedding. Moreover, there is no direct mechanism to assign fluid channels to component ports after expansion. Our solution to these issues is to simultaneous compute a port assignment and a new set of fluid channel routes that remains planar in the presence of expanded components.

2.4.2 Shift Expansion

The Shift Expansion (ShiftEx) method tries to reduce the amount that each components shifts the set of other components surrounding it. Let c_i be the component currently being expanded, and assume that C contains only those components that have not yet be expanded. The basic premise is to shift the position of component $c_j \in C$ by an amount that is proportional to the distance between c_j and c_i in the x- and y-directions, which moves the component c_j out of the expansion area of c_i with a smaller shifting factor than in [38]. To do this, shift expansion computes shift factors d_x and d_y which are applied to each component c_j, shifting it to position $(x_j + |x_j - x_i| \times d_x, y_j + |y_j - y_i| \times d_y)$; the shift may include an additional term, Δ_{buf}, to add additional spacing if routability is a concern.

The position of component c_i is represented by coordinate (x_i, y_i) at its upper left corner; the length and width of c_i after expansion are l_i and w_i respectively. To compute d_x and d_y (Figure 2.7a and Figure 2.7b respectively) the algorithm selects three not-yet-expanded components, c_f, c_g, and c_h, which are the points lying closest to c_i in the respective regions above, inside, and to the left of c_i's expanded component. The algorithm scales length l_i and width w_i by the differences in the x- and y- coordinates between c_i and the three selected points, yielding terms $d_{x,f}$, $d_{x,g}$, and $d_{x,h}$ in the x-direction, and $d_{y,f}$, $d_{y,g}$, and $d_{y,h}$ in the y-direction; d_x and d_y are then selected as the respective maximum values between the two sets of three terms and the components are sifted by the same factor x- and $y-$ directions (Figure 2.7c).

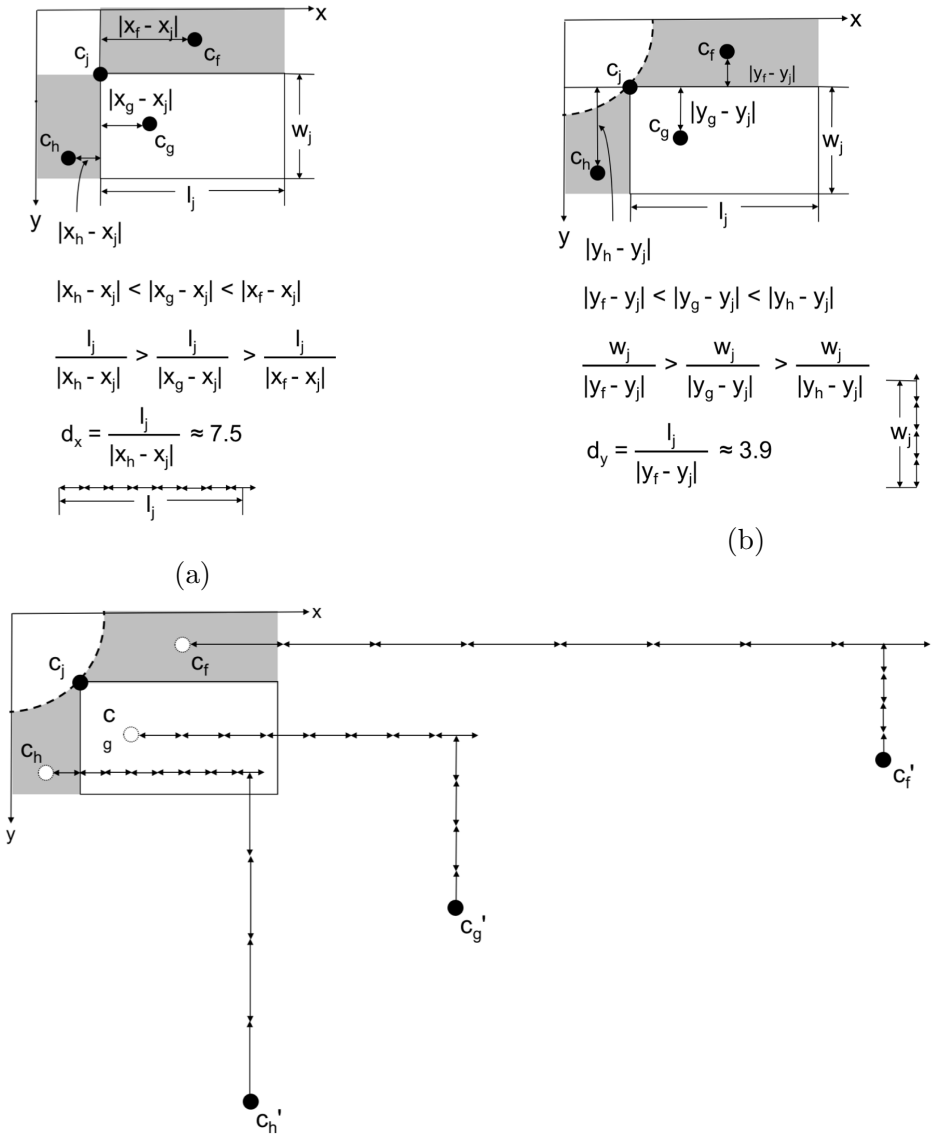


Figure 2.7: Shifted Expansion example: (a)-(b) Computation of d_x and d_y; (c) each non-expanded point is shifted based on d_x, d_y and its distance from the expanding component

2.4.3 Scaled Expansion

The Scaled Expansion (ScaleEx) expands upon ShiftEx and tries to find the smallest global scale factor that can remove component overlap from the initial placement. The algorithm

checks each possible integer scale factor in the x- and y-dimensions until a valid placement (including any buffer spacing) is found.

Scaled expansion begins with integer scale factors of 2 in the x- and y-dimensions. For component $c_i \in C$, the algorithm multiplies x_i by the x-dimension scaling factor, $scale_x$ and y_i by the y-dimension scaling factor, $scale_y$. The algorithm then determines if any two components overlap. If so, the algorithm reverts each component c_i back to its initial location, (x_i, y_i). If a valid placement is not found in either the x- or y-dimension, the algorithm increments the scaling factor(s) and repeats the process until no overlap occurs.

2.5 DIagonal Component Expansion (DICE)

Here we introduce Diagonal Component Expansion (DICE), which improves upon the original component expansion algorithm outlined in Section 2.3. As it's name suggests, DICE tends to place components on a diagonal axis from the upper left corner of the chip to the lower right corner, yielding a compact, yet routable, layout.

2.5.1 Component Selection via Circular Propagation

DICE selects components one-by-one for expansion, expanding each point into a two-dimensional component. Components are processed in *Circular Propagation* order, as shown in Figure 2.8. The origin, $(0, 0)$, is the upper left corner. Components are expanded in non-decreasing order of their distance from the origin. Equidistant components lie on a circle centered at the origin; as a tiebreaker, equidistant components are processed in increasing order of their y-coordinates.

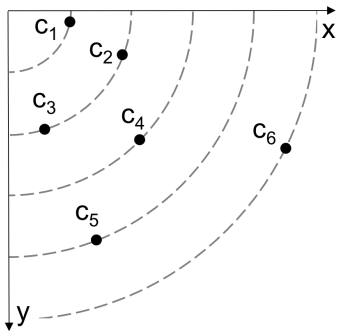


Figure 2.8: Component selection by circular propagation. Components are selected in increasing order of subscript.

2.5.2 Diagonal Expansion

Require: $C :=$ set of components in the system, $\Delta_{buf} :=$ minimum component spacing
Ensure: All $c_i \in C$ placed with no overlap

for all $c_i \in C$ **do**
  $C \leftarrow C \setminus \{c_j\}$
  $\Delta_x \leftarrow 0$, $\Delta_y \leftarrow 0$
  $c_i.expand\_component()$
  for all $c_i \in C$ **do**
    if $c_i.inside\_left\_or\_above(c_j)$ **then**
      $\delta_x \leftarrow x_i + l_i - x_j$
      $\delta_y \leftarrow y_i + w_i - y_j$
      if $\delta_x > \Delta_x$ **then**
        $\Delta_x \leftarrow \delta_x$
      end if
      if $\delta_y > \Delta_y$ **then**
        $\Delta_y \leftarrow \delta_y$
      end if
    end if
  end for
  for all $c_i \in C$ **do**
    $x_i \leftarrow x_i + \Delta_x + \Delta_{buf}$
    $y_i \leftarrow y_i + \Delta_y + \Delta_{buf}$
  end for
end for

Figure 2.9: Pseudocode for the DICE method

Diagonal Expansion (pseudocode in Figure 2.9) tries to minimize the necessary increase in chip area used during the component expansion step of a planar embedding-based placement method by shifting components across the device diagonal.

Let c_j denote the component selected for expansion. DICE calculates a *shift factor* in the x- and y-directions for each component c_i in the regions inside, above, or to the left of c_j's expanded two-dimensional area. The shift factor in the x-direction, $\delta_x = x_j + l_j - x_i$, is the distance between c_i's x-coordinate and c_j's right edge (Figure 2.10a); the shift factor in the y-direction, $\delta_y = y_j + w_j - y_i$, is the distance between c_i's y-coordinate and c_j's bottom

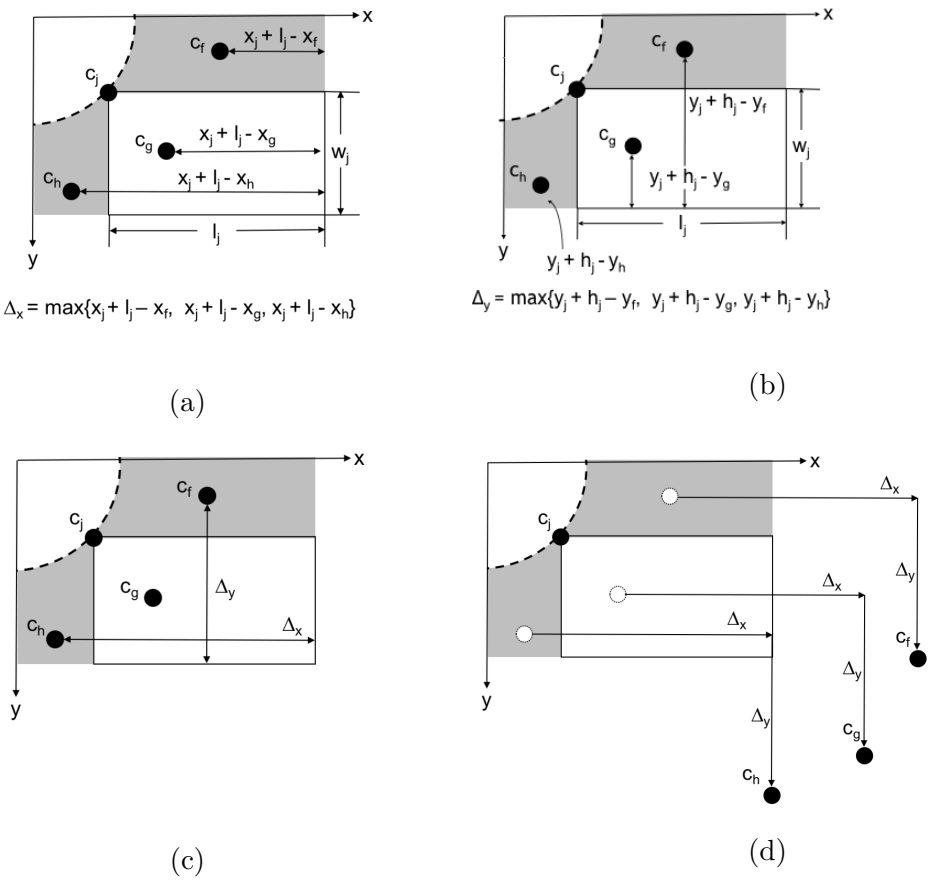


Figure 2.10: DICE example: (a)-(c) Computation of Δ_x and Δ_y; (d) each non-expanded point is shifted accordingly.

edge (Figure 2.10b). DICE takes the maximum calculated δ_x and δ_y as the shift factors in the x and y directions, Δ_x and Δ_y (Figure 2.10c). The last step is to reposition components to remove overlap. Each remaining component in $c_k \in C, k > j$ is shifted to the right by Δ_x and downwards by Δ_y (Figure 2.10d). If routability is a concern, a constant Δ_{buf} can be added to Δ_x and Δ_y to add extra buffer space to assist the fluid channel router. The new coordinate for component c_i is $(x_i + \Delta_x + \Delta_{buf}, y_i + \Delta_y + \Delta_{buf})$. These shifts will spread the components along the device diagonal, leaving the majority of the component's ports unblocked by other components and free for use in routing.

2.6 Network-flow Based Routing

2.6.1 Routing Grid

The next step is to instantiate a routing grid $R = (U, F)$, where U is a set of grid points, and F is a set of edges representing potential channel routes between adjacent grid points. For each component $c_i \in C$ a vertex u_i for the ports $p_h \in P_i$ is instantiated and added to U. A grid of vertices is then instantiated in the empty space between components. Pseudocode is presented in Figure 2.11. In lines 17 and 20, edges that represent potential routing channel segments are added to F by instantiating a bidirectional edge f_i with a capacity of 1 between $u_i \in U$ and $u_j \in U$ if and only if $(u_j.x - u_i.x == 1) \oplus (u_j.y - u_i.y == 1)$.

The network flow model ensures that no edge is used more than once. It is also necessary to ensure that no vertices are used more than once in routing. This is accomplished by splitting each vertex $u_i \in U$ into u_i' and u_i'' and adding a directed edge $f_i = (u_i', u_i'')$ to

Require: $C :=$ set of all components in the system
Require: $max\_x, max\_y :=$ the maximum x and y values in the plane
Ensure: $R := (U, F)$ grid of vertices

```
1: for all c_i ∈ C do
2:    for all p_h ∈ c_i do
3:       U ← U ∪ u_i = (p_h.x, p_i.y)
4:    end for
5: end for
6: for all 0 < x < max_x do
7:    for all 0 < y < max_y do
8:       if !within_component(x, y) then
9:          U ← U ∪ u_i = (x, y)
10:      end if
11:   end for
12: end for
13: for all 0 < x < max_x do
14:   for all 0 < y < max_y do
15:      u_i ← (x, y)
16:      F ← F ∪ get_east_neighbor(u_i)
17:      F ← F ∪ get_south_neighbor(u_i)
18:   end for
19: end for
```

Figure 2.11: Pseudocode for the grid creation algorithm used in Network-flow based routing

F. All incoming edges to u_i are now forced through u_i', and all outgoing edges from u_i leave through u_i''. Thus, any fluid channel that routes through u_i, must go through the edge f_i, and the capacity constraint on edges ensures that there can be at most one such channel using that vertex.

2.6.2 Network Flow Model

Once the grid $R = (U, F)$ has been constructed, the next step is to route channels between the components. This is accomplished using a network flow routing method based on Ref. [69]. Components are processed in-order, and unrouted channels that are incident on each component are routed together using this model. The special nodes super sources, super

Require: A grid $R = (U, F)$ representing all available routing space
Ensure: A network flow preparation from $c_i \in C$ to all $t_j \in T_i$

1: $U \leftarrow U \cup u_{supersink}$
2: **for all** $t_j \in T_i$ **do**
3: $\quad U \leftarrow U \cup u_{sinkgroup\_t_i}$
4: $\quad F \leftarrow F \cup f_j = (u_{sinkgroup\_t_i}, u_{supersink}, cap = 1, cost = 1)$
5: $\quad$ **for all** $p_k \in P_j$ **do**
6: $\quad\quad U \leftarrow U \cup u_{p_k}$
7: $\quad\quad F \leftarrow F \cup f_{p_k} = (u_{sinkgroup\_t_j}, u_{p_k}, cap = 1, cost = 0)$
8: $\quad$ **end for**
9: **end for**
10: $U \leftarrow U \cup u_{supersource}$
11: **for all** $p_j \in P_i$ **do**
12: $\quad U \leftarrow U \cup u_{p_j}$
13: $\quad F \leftarrow F \cup f_{p_j} = (u_{supersource}, u_{p_j}, cap = 1, cost = 0)$
14: **end for**

Figure 2.12: Pseudocode for adding route enforcement nodes for the Network-flow based routing method

sinks and sink groups; are added to the routing problem which enables the network flow to simultaneously perform port assignment as well. Figure 2.12 shows the pseudocode for adding preparing the routing grid for network flow routing and Figure 2.13 a visualization of the same process.

For routing a component $c_i \in C$ set of routes from c_i to all $t_j \in T_i$ is found by computing the maximum flow from $u_{supersource}$ to $u_{supersink}$, followed by a path reclamation step adapted from Lee's algorithm [33]. The paths computed by the network flow algorithm include port assignment at the source and sinks, and may present multiple valid paths. The purpose of the trace back, as shown in Figure 2.14 is to compute the shortest valid path from the port p_k at each sink t_i to its corresponding port p_j at the source component c_i, as determined by the solution to the network flow problem. The super source, super sink, and

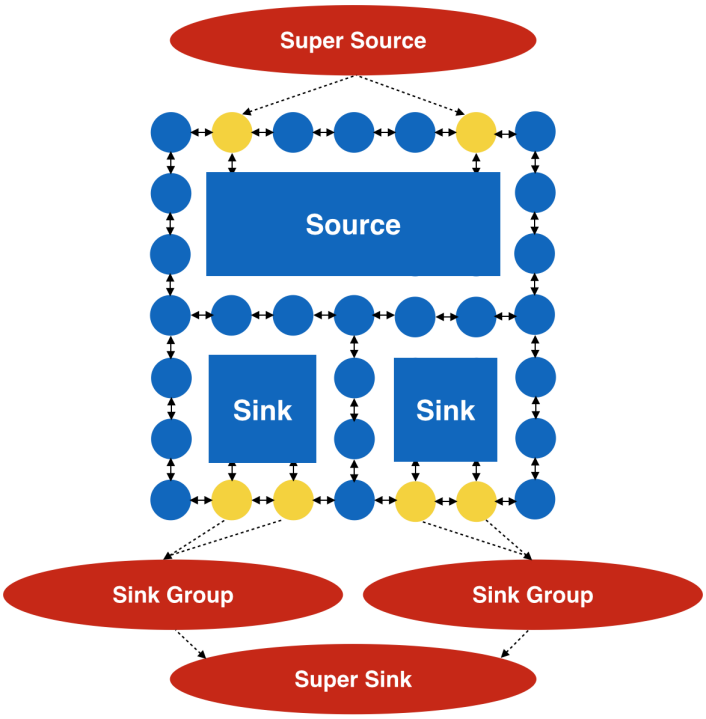


Figure 2.13: The addition of the super source, super sink, and sink group vertices with accompanying edges allows the use of minimum cost maximal flow algorithms to do both port selection and channel routing. Note that a sink group is connected to every port of a particular sink

sink groups along with their accompanying edges are then removed from the routing grid, and the process repeats for each component in the system, taking care not to route fluid channels that have already been routed.

One problem that may occur is that a route between components c_i and c_j may abut a third component, c_k, potentially blocking one of its ports. To avoid this, we create buffer zones of a few vertices around every component that is not currently being routed. The vertices within the buffer zone are removed from the routing grid ensuring that the

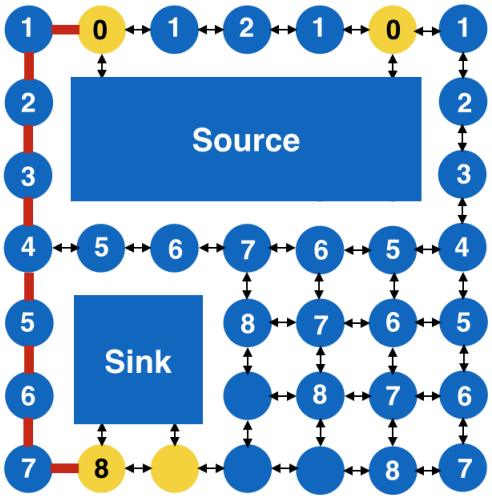
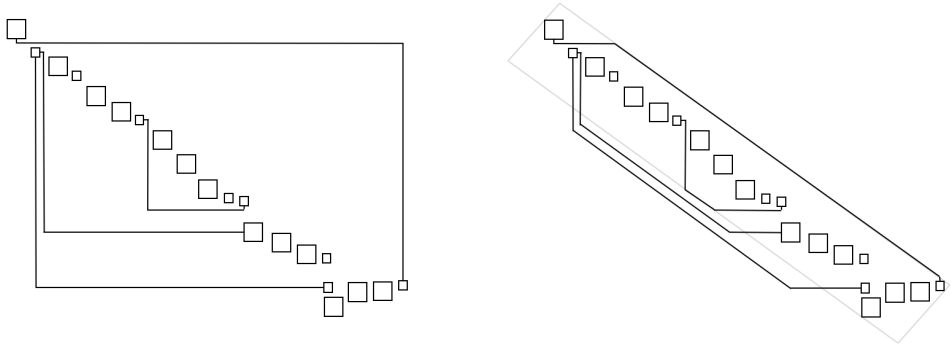


Figure 2.14: The maximum flow minimum cost network flow algorithm numbers the nodes as they are discovered, ending when it reaches an unused port. These numbers are then used to trace back the path of the channel route.

ports are not blocked, and only added back to the grid temporarily while routing to or from the respective component.

As fluid channels are routed one-by-one, the routing grid becomes fractured, leading to failures due to routed channel intersections. If a routing failure occurs, the old routes are removed and the queue of components is reordered so that components whose routes have failed in the past are now routed first. We limit the number of times that the component queue may be reordered; if this limit is exceeded, we declare a routing failure. No routing failures were observed in our experiments; however, they may still occur in practice. In the presence of failure, the mVLSI flow layer architecture must be redesigned.

2.6.3 Diagonally-constrained Channel Routing



(a) (b)

Figure 2.15: An mVLSI netlist laid out using DICE with (a) unconstrained routing; and (b) diagonally constrained routing. Only routes that cross the diagonal boundaries are depicted.

Figure 2.15a shows a legal fluid channel routing solution following DICE; only the routes that cross the boundary of the diagonal envelope are shown. These routes are minimum length under the assumption that only horizontal and vertical directions may be used; however, they unnecessarily extend the chip area.

Diagonally-constrained routing re-routes the fluid routing channels that cross l_1 and l_2. As shown in Figure 2.15b, the new routes run parallel to l_1 and l_2 while obeying spacing rules. If diagonal routes are not allowed, which they generally are in most microfluidic fabrication techniques, these routes can be approximated using a zig-zag pattern. The expansion in chip area depends on the number of re-routed channels, channel width, and foundry-imposed spacing rules between channels.

Since the mVLSI layout is planar, no edges that cross the diagonal chip boundaries overlap, and each crossing edge crosses twice. It suffices to order the crossing edges by the

positions of their crossing points along the chip boundary; they can be routed optimally using the Left Edge Algorithm [21]. From there, the resulting mVLSI chip can be cut and rotated. The high degree of routability provided by diagonal expansion, coupled with its ability to re-route channels along the diagonal axis, reduces the space needed to fully place and route a chip. This tends to improve area utilization and decreases reduce flow channel route length.

However, it is possible for a placement to be generated that cannot yield a valid channel routing; either while routing the flow layer, or when routing the control layer. When either the flow or control layer fails to route then the current partial route is removed, the constraints used to generate the original placement are relaxed, and the netlist is re-placed. This process can then repeat as necessary until the flow and control layers are validly routed.

2.7 mVLSI Placement Metrics

Although mVLSI technology and its underlying physical design processes share many principle similarities with semiconductor VLSI, there are also important differences which cannot and should not be ignored. Prior work has mistakenly evaluated the quality of mVLSI layouts using semiconductor metrics such as area and wirelength (fluid channel length) as proxies for good quality layouts. Although we report these metrics in the next section for the purpose of enabling direct comparison with prior work, we do so with reservations. Without considering the larger context, these metrics are fundamentally flawed, as they do not account for the factors that influence performance (bioassay execution time) or the

differences between semiconductor and mVLSI fabrication processes, economies of scale, etc. We dive deeper into these topics in Chapter 6.

2.7.1 Area

In semiconductor VLSI, chip area correlates directly to cost (number of chips per wafer) and indirectly to performance (reducing area may, in some cases, reduce the lengths of the longest wires routed on-chip). While there are a number of different methods for fabricating microfluidic devices, most lack the economies of scale that are present in semiconductor manufacturing. In an academic setting, the most labor-intensive steps are alignment, hole punching, and testing, which are often performed by PhD students or postdocs. If a fixed number of chips (say 100) need to be fabricated and tested to produce statistically robust results for a publication, then the cost driver is not the number of chips per wafer but the manual labor involved. Thus, area minimization (within reason) is far less important than producing a functionally correct layout. These issues have also hampered industrial adoption of mVLSI technology; industrial preference is strongly biased toward passive devices (no valving) using fabrication processes such as injection molding or glass etching. The purpose of this statement is *not* to disparage academic efforts on mVLSI design automation; it is simply to place the work in its appropriate context.

The number of mVLSI chips per mold depends on wafer size (mold size) and chip size. For large and complex mVLSI chips, the number of chips per mold may be relatively small (e.g., 10 or less); these are, of course, the most challenging chips to lay out algorithmically. For a given chip design, a significant reduction in area through more effective physical design could, in principle, free up enough space to add another chip to the mold; on the

other hand, incremental reductions in area that do not increase the number of chips per mold will simply reallocate area from the device to the extra material that is removed and discarded. As placement algorithms become increasingly effective, incremental improvements in the $1-2\%$ range (which would certainly be valuable in semiconductor VLSI) are likely to have minimal impact, outside of rare corner cases. Thus, it is fair to question the utility and practicality of long-running and optimal and near-optimal algorithms such as those based on integer linear programming (ILP) [59] or boolean satisfiability (SAT) solving [17].

2.7.2 Routing Channel Length

In semiconductor VLSI, channel length can directly affect clock frequency, power dissipation and signal integrity; these are non-issues in mVLSI. mVLSI chips are not aggressively clocked; they do not consume power directly, as fluid is driven by external pressure sources, which are typically plentiful in biological laboratories; and fluid transport integrity issues are minimized due to pumping. This issue is much more challenging and prevalent for passive device designs, which do not utilize pumps for the movement of fluid and where reductions in channel length may be more useful.

Reducing fluid channel length can reduce fluid transport times; however, in microfluidics, bioassay execution time is typically dominated by biological phenomena (e.g., culturing cells), and in any given scenario, the biological phenomena may or may not be dominant when considering the entire end-to-end workflow of the laboratory. Thus, the performance motive to shave a few seconds from a process that may take hours or days is

questionable at best. Any claim that reducing fluid channel length is integral to mVLSI chip performance is spurious.

There is, however, one benefit that can be accrued by reducing fluid channel lengths. The key issue is that fluid is transferred in continuous flows, not discrete packets. Thus, reducing fluid channel lengths can reduce the total volume of fluid required to perform a bioassay. This can lead to tangible cost savings when dealing with limited sample volumes and expensive reagents. Additionally, shorter channels generate less resistance and therefore require less pressure to drive. While this is rarely a problem for active devices, it can be a limiting factor in passive devices that may be driven with head pressure.

2.8 Experimental Results

DICE was implemented in C++ using a *unitless* grid, which decouples the layout and design rule checking processes from the manufacturing resolution of any one specific mVLSI technology [42]. From the layout, we can easily count the number of switches inserted (if applicable), measure area (and/or area utilization) and (gridless/normalized) fluid channel length, and convert the resulting layout to a technology-specific grid. From there, we can lay out one or more instances of an mVLSI chip (or a heterogeneous set of chips) on a silicon wafer/mold of a known size. In all cases, the Network-flow based router introduced in Chapter 2 is used to perform the routing step.

2.8.1 Experimental Comparison

We compare six different mVLSI layout algorithms, including two variants of DICE. We briefly summarize these methods here.

- **Simulated Annealing (SA)** refers to the simulated annealing algorithm proposed by McDaniel et al. [41] and introduced in Chapter 2, which uses Hadlock's Algorithm for channel routing. SA cannot guarantee a planar layout for planar netlists; all other methods included here provide this guarantee.

- **Baseline Expansion (BaseEx)** refers to the Baseline component expansion method (Section 2.4.1).

- **Shift Expansion (ShiftEx)** replaces the BaseEx component expansion step with Shift Expansion (Section 2.4.2).

- **Scaled Expansion (ScaleEx)** replaces the BaseEx component expansion step with Scaled Expansion (Section 2.4.3).

- **Diagonal Component Expansion Unconstrained (DICE-U)** replaces the BaseEx component expansion step with DICE (Section 2.6.3), implemented with unconstrained fluid channel routing (Figure 2.15a).

- **Diagonal Component Expansion (DICE)**, implemented with diagonally-constrained fluid channel (Figure 2.15b).

2.8.2 Benchmarks

We make our comparison using netlists for four real-world planar mVLSI chips that have been designed, fabricated, and evaluated in literature, as well as five netlists obtained by synthesizing synthetic benchmarks. The real-world netlists are as follows:

- **aquaflex-3b & aquaflex-5a:** proprietary mVLSI device netlists provided by Microfluidic Innovations, LLC.
- **hiv:** a multi-layer polydimethylsiloxane (PDMS) chip that performs a bead-based HIV1 p24 sandwich immunoassay [34].
- **mgg:** a molecular gradients generator that can generate five concentration levels of a two-sample mixture [52].

The five synthetic benchmarks were generated by compiling a set of publicly available DAG specifications through an established mVLSI architectural synthesis flow [45, 39, 22]. Additional information for these benchmarks are presented in Section 6.3.2 and Table 6.2. Experiments were run using a buffer of 5 grid spaces for each component. Legal planar mVLSI embeddings were obtained for all component expansion algorithms.

2.8.3 Results and Analysis

For each component expansion heuristic and benchmark, we report the area utilization (Figure 2.16) the ratio of component area to total chip area expressed as a percentage) and the average routing channel length (Figure 2.17) and discuss channel intersections.

Computation time of the framework is dominated by the routing and port assignment phase; varying the component expansion heuristic made a negligible impact.

Channel Intersections

As was stated in Chapter 2, channel intersections necessitate the introduction switches, which in turn, require additional control lines; these, in turn, increase chip area, and may lead to a design rule violation of the number of switches in the netlist exceeds found-specific limits.

The heuristics based on planar placement with component expansion, as expected, did not introduce any new intersections, given that the input netlists were planar; SA, in contrast, introduced numerous unnecessary channel intersections to all netlists (¿10 intersections in all cases). Consequently, few, if any, of the netlists produced by SA could be fabricated at the Stanford Microfluidics Foundry [2], which limits the number of I/O hole punches to 35; the total number of punches would be the sum of the netlist's initial fluidic I/O and control requirement, plus two additional control lines per intersection. These results demonstrate the need to properly account for planar embedding during layout.

Area Utilization

Figure 2.16 reports the area utilization (i.e., the percentage of total chip area dedicated to components and channels). DICE achieved the highest area utilization for each benchmark, and the gap between DICEs result and the best result of the remaining heuristics was significant in all cases. On average, DICE improves area utilization by a factor of $8.90x$

compared to BaseEx and 2.64x compared to ScaleEx. SA does not perform particular well in this comparison, except for the Synthetic-5 benchmark, where it achieves the second highest area utilization; however, this result is built on top of 207 channel intersections (necessitating 414 control lines), which cannot be fabricated.

Figure 2.15 illustrates the reason that DICE improves area utilization compared to DICE-U. DICE-U performs routing on a square chip with a relatively long and densely packed (in terms of components) diagonal; it finds minimum-Manhattan distance fluid channel routes, which mostly follow simple X-Y and Y-X routing patterns with one bend. In contrast, DICEs diagonally constrained routing tends to reduce overall chip area. ScaleEx, in contrast, tends to generate chips with shorter diagonals than DICE-U, leading to smaller rectangular area and higher area utilization; however, this inhibits the effectiveness of

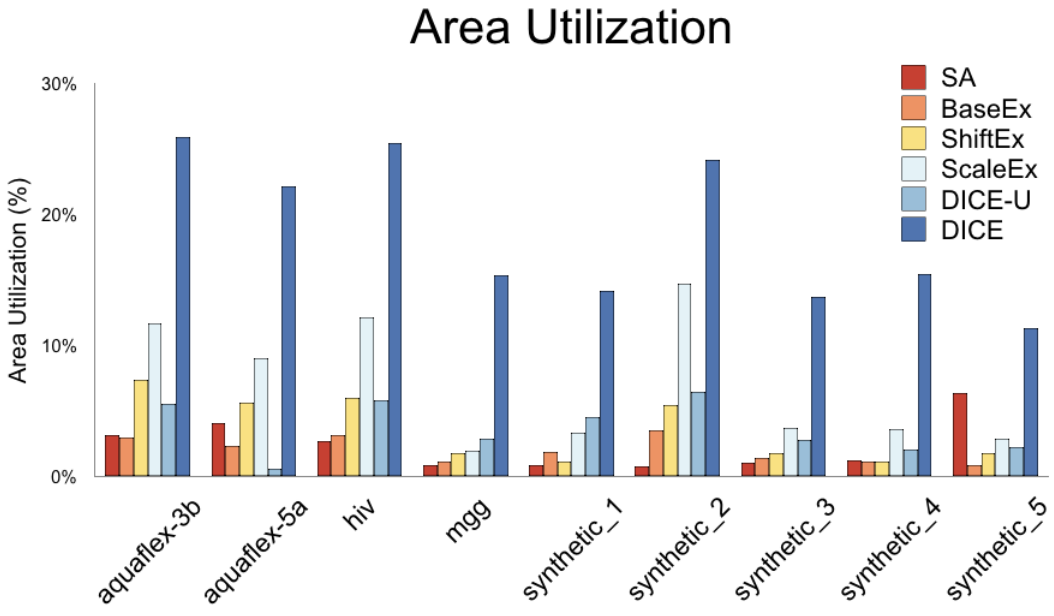


Figure 2.16: Comparison of area utilization between SA, BaseEx, ShiftEx, ScaleEx, DICE-U, and DICE

diagonally-constrained routing. These observations explain why DICE achieves the highest area utilization reported in Figure 2.15.

Average Channel Length

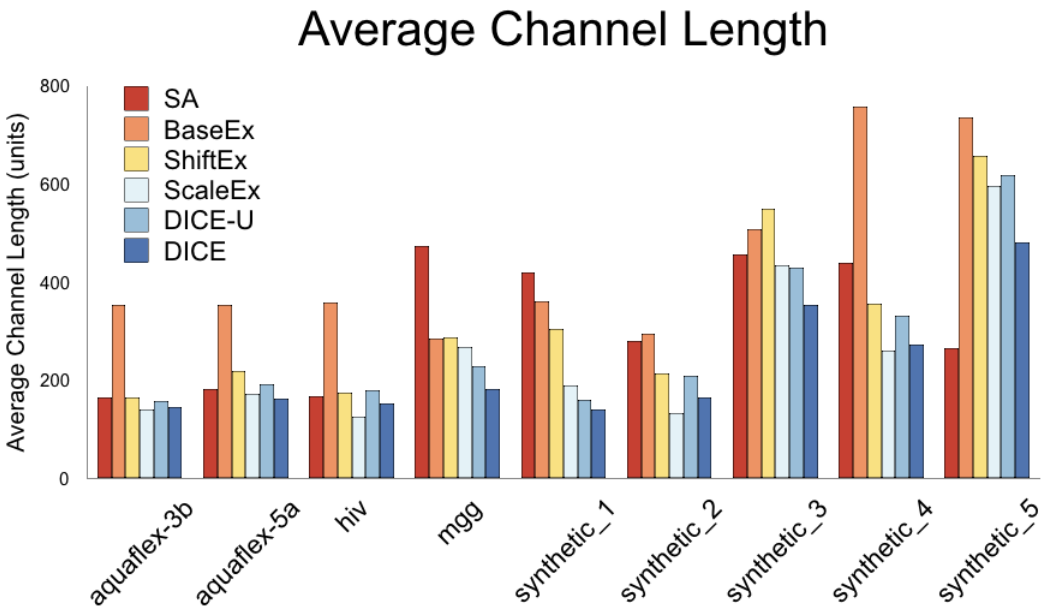


Figure 2.17: Comparison of the average channel length between SA, BaseEx, ShiftEx, ScaleEx, DICE-U, and DICE

For all benchmarks in Figure 2.17, either ScaleEx or DICE achieve the shortest average fluid routing channel length, and in most cases, the disparity between the two is quite small. These results indicate that the initial planar embedding solution is quite effective in terms of limiting the fluid channel length, and ScaleEx retains those benefits by maintaining the same relative position of components. DICE, in contrast, repositions components in a manner that primarily improves chip area while retaining the channel

length benefits of the planar embedding. On average, DICE improves average fluid routing channel length by 47.4% compared to BaseEx and 9.62% compared to ScaleEx.

Runtime

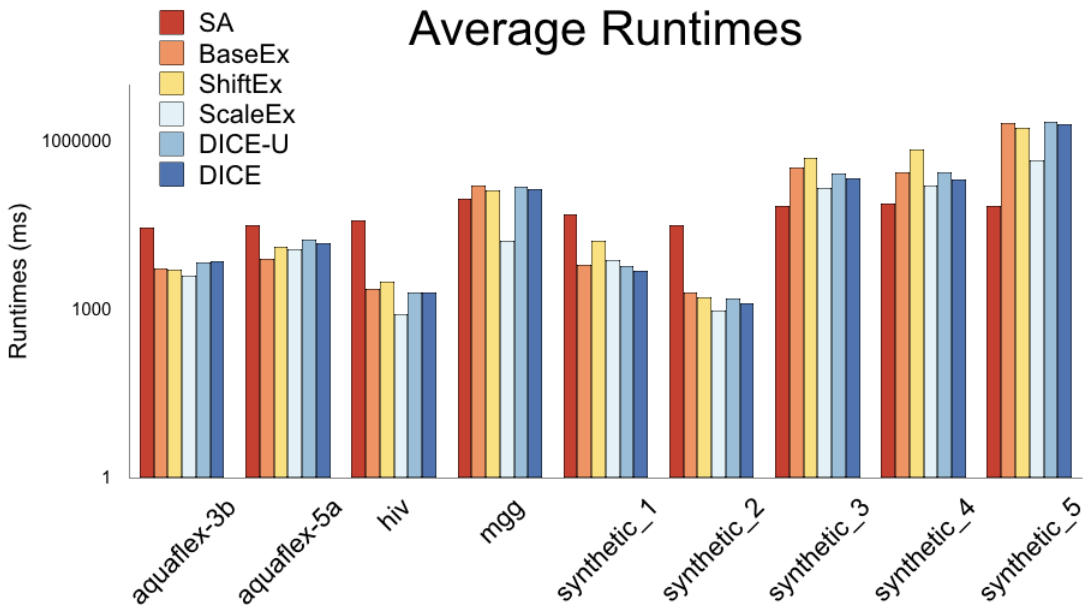


Figure 2.18: Comparison of runtimes between SA, BaseEx, ShiftEx, ScaleEx, DICE-U, and DICE

Figure 2.18 reports the runtimes of the planar layout algorithms. The runtime of SA, it should be noted, depends on parameter configurations, and is thus variable. We used the same SA parameter settings as in Ref. [41], which coincidentally had SA running approximately as fast as the planar embedding methods that we evaluated here. It is also worth noting that SA uses a router based on Hadlock's Algorithm, as opposed to the network flow-based router used by the planar embedding heuristics [69]. Better results, in

principle, could be obtained by letting SA run longer; that said, it seems unlikely that SA would achieve planar layouts within a reasonable runtime.

The remaining placers typically complete in milliseconds; the router dominates the total runtime. For a given benchmark, variations in runtime among the different layout heuristics is determined primarily on how quickly the router can obtain a valid solution. Future work may attempt to reduce the runtime by exploring more efficient routing algorithms.

2.8.4 Case Study: aquaflex-3b

As a case study, Figure 2.19 shows the flow layers of the aquaflex-3b benchmark using all six placement and routing heuristics. SA (Figure 2.19a) yields a non-planar layout with poor area utilization; in principle, adjusting the parameters to provide a longer runtime could yield better results, however, it is unlikely to guarantee a planar layout that could be fabricated.

BaseEx (Figure 2.19b) achieves a planar layout, which can be fabricated, but with poor area utilization. Each time BaseEx expands a new component, it shifts the positions of all components that have not yet been expanded by the expansion amount in the horizontal and vertical directions. For example, consider two points on a common vertical axis that would not overlap if expanded. When the first point is expanded, the second will be shifted in both directions, arguably unnecessarily. This ensures that any horizontal or vertical line cutting through the design will intersect at most one component. Although BaseEx preserves planarity, it does so at the expense of area utilization.

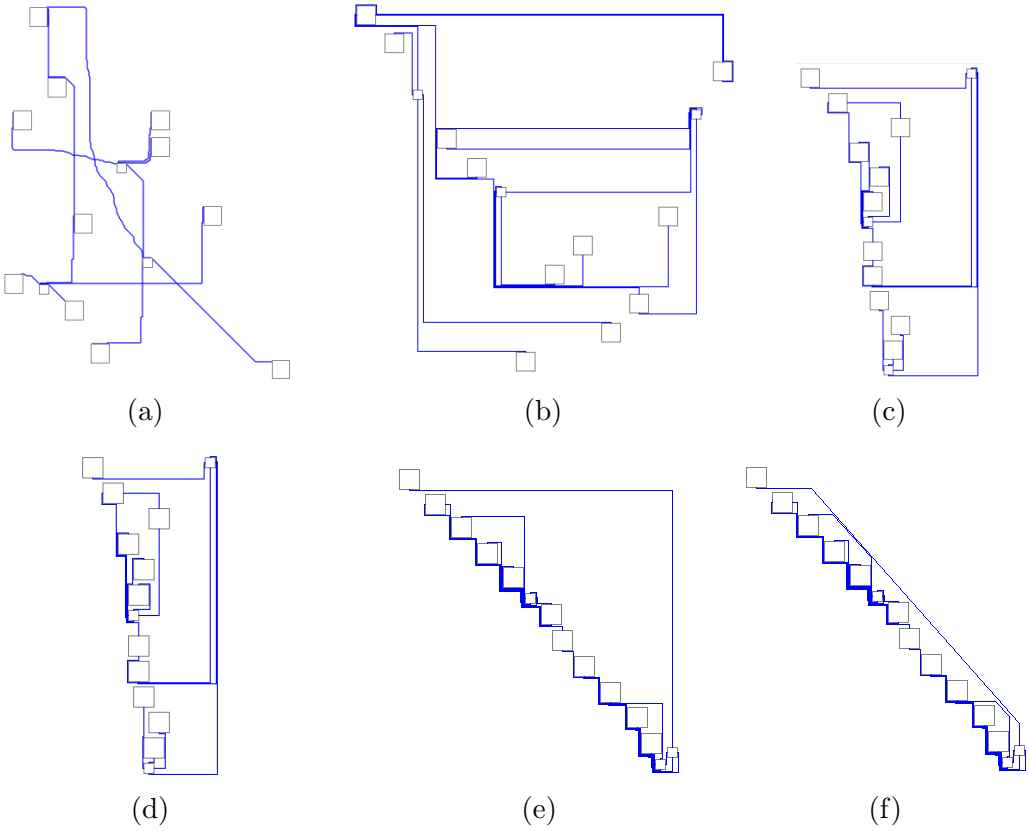


Figure 2.19: Visual comparisons of layouts for the AquaFlex-3b device using (a) SA, (b) BaseEx, (c) ShiftEx, (d) ScaleEx, (e) DICE-U, and (f) DICE

ShiftEx (Figure 2.19c) and ScaleEx (Figure 2.19d) generate similar layouts, with the latter achieving slightly better area utilization. The key to the improvement is to scale the length of the horizontal and vertical shifts by the distance of the component being expanded to its nearest neighbors, which yields shorter shift distances than BaseEx. ShiftEx retains a slight advantage over ScaleEx because it computes a shift distance independently for each not-yet-expanded component, while ScaleEx computes one scale factor that is applied to shift all not-yet-expanded components.

DICE-U (Figure 2.19e) achieves a tighter layout by shifting components in a manner that tends to lay them out along a diagonal axis. Although components are clustered along the diagonal, the length of the diagonal and total chip area is larger than the results produced by ShiftEx and ScaleEx, and may result in long fluid routing channels whose length is equal to the Manhattan Distance between their incidental components. DICE (Figure 2.19f), which allows for diagonal routing parallel the diagonal axis, eliminates these inefficiencies, once the chip is cut out and rotated.

SA (Figure 2.19a) does not achieve a planar layout because the Hadlocks-based router that it utilizes does not explicitly try and avoid intersections. However, because this method does not try and improve upon a initial placement that is planar or close to planar, even when it is paired with a router that disallows intersections explicitly it is unlikely to generate a layout that can be routed. The implementation of SA used here is based directly from the original publication [41], and uses the values suggested for temperature and movements per degree. This implementation is highly dependent on the initial placement because it relegates inputs and outputs to the edges of the device with no mechanism to move move those edges closer to the center. While we present a standard implementation here, in Chapter 3 we implement SA with additional mechanics to overcome these issues and increase the temperature and movements per degree while still allowing it to introduce an infinite number of intersections. This produces an algorithm that can be thought of as near-optimal, since it is allowed to concentrate on minimizing route lengths and area without generating a valid layout. Comparisons to the BaseEx and DICE methods are presented in Chapter 3 for completeness.

2.9 Conclusion

The Planar Placement algorithms presented here are among the first attempts to automatically generate a flow-layer without inserting any additional intersections, allowing them to be used for both active and passive devices. While these methods do not guarantee that such a design can be found, the empirical analysis in this chapter, as well as more extensive benchmarking described later in Chapter 6, shows that they perform well in scale and for a wide variety of netlist topologies. The Planar Placement algorithms work very well in the general case, however we have identified many netlist topologies for which they exhibit high failure rates, and others for which dramatic improvements can easily be identified. We further explore the effects of netlist topology on these methods in Chapter 6 and in Chapter 3 we introduce new microfluidic physical design techniques that are tied to common netlist topologies, and leverage this insight to drive design improvement.

Chapter 3

Directed Placement for mVLSI Devices

3.1 Introduction

After the development of Planar Placement and Network-flow based routing, a number of new methods were introduced which also focused on created a planar layout that avoided intersections. These new methods were primarily developed using integer linear programming (ILP) [59] and boolean satisfiability (SAT) [17] techniques to yield a (near-) optimal solution for both the flow and control layers. While these techniques yield layouts with relatively small device area and channel lengths, both ILP and SAT methods have difficulty scaling to large problem sizes, meaning that as microfluidic very-large-scale integration (mVLSI) devices scale up to larger numbers of components and connections, these methods are unlikely to scale.

Rather than creating an ILP or SAT solution to this problem, we developed a new method based on observations of how domain experts were creating their device designs. After reviewing devices from literature and discussing techniques with domain experts, the realization was made that most devices are laid out linearly in a stepwise fashion. While this is not a universal rule, designs primarily have fluidic inputs on one end and fluidic outputs on the other with control I/O being placed around the perimeter (often near the top and bottom of the device). Between the inputs and outputs, a number of process steps are laid out in a linear manner. Through this observation, we created a method that utilizes input and output ports as "beginning" and "ending" points, respectively. The netlist is then represented as a graph and traversed from the beginning to the end, with each component's distance from the beginning representing which process step it belongs to and where it will be laid out linearly from the input ports.

3.2 Preliminaries

An mVLSI netlist $M = (C, E)$ consists of a set of components, C, and a set of edges, E, between them. A component $c_i \in C$ is a tuple $c_i = (T_i, P_i, x_i, y_i, h_i, w_i)$ where T_i is the set of neighboring components to c_i, P_i is the set of c_i's ports, (x_i, y_i) is the coordinate location of the upper left corner of c_i, and h_i and w_i are the height and width of c_i, respectively. A port on a component $c_i \in C$, $p_{i,j} \in P_i$ is located at $(a_{i,j}, b_{i,j})$, a point on the perimeter of c_i; c_i is called a *terminal component* if $|T_i| = 1$. An edge, $e_i \in E$, is a pair of components $e_i = (c_i, c_j)$ which represents a fluidic connection between them. An optional set of components $I \subset C$ can also be provided that represents the inputs of the microfluidic device.

A lane L_i is defined to be an ordered set of components that align vertically. These lanes are numbered and ordered $L_0, L_1, ...L_n$, where L_0 is the leftmost lane and L_n is the right most lane. The first component in the set $c_0 \in L$ is at the top of the lane and the last component in the set $c_{|L|-1} \in L$ is at the bottom of the lane. Adjacent lanes may be separated by an optional buffer space Δ to improve routability and/or to satisfy fabrication design rules relating to spacing.

3.3 Placement

3.3.1 Preprocessing

Directed Placement uses a microfulidic netlist as an input, but does not require a microfluidic application in order to perform placement and routing. Because no application is given as input no optimizations can be made to the netlist, since it would be impossible to determine if a change to the netlist would render the application unable to map. Previous methods for generating and optimizing netlists based on applications [45] have been proposed, and methods to optimize the netlist before placement and routing are compatible with the Directed Placement method. For these reasons architectural optimization is out of scope for this work, and the assumption is made that all components and connections are required to create a valid layout.

Directed Placement, like Planar Placement, requires that the input device architecture is planar as this is a requirement for the manufacturing of the physical device. Planarity in a graph can be determined by the absence of the Kuratowski subgraphs K_5

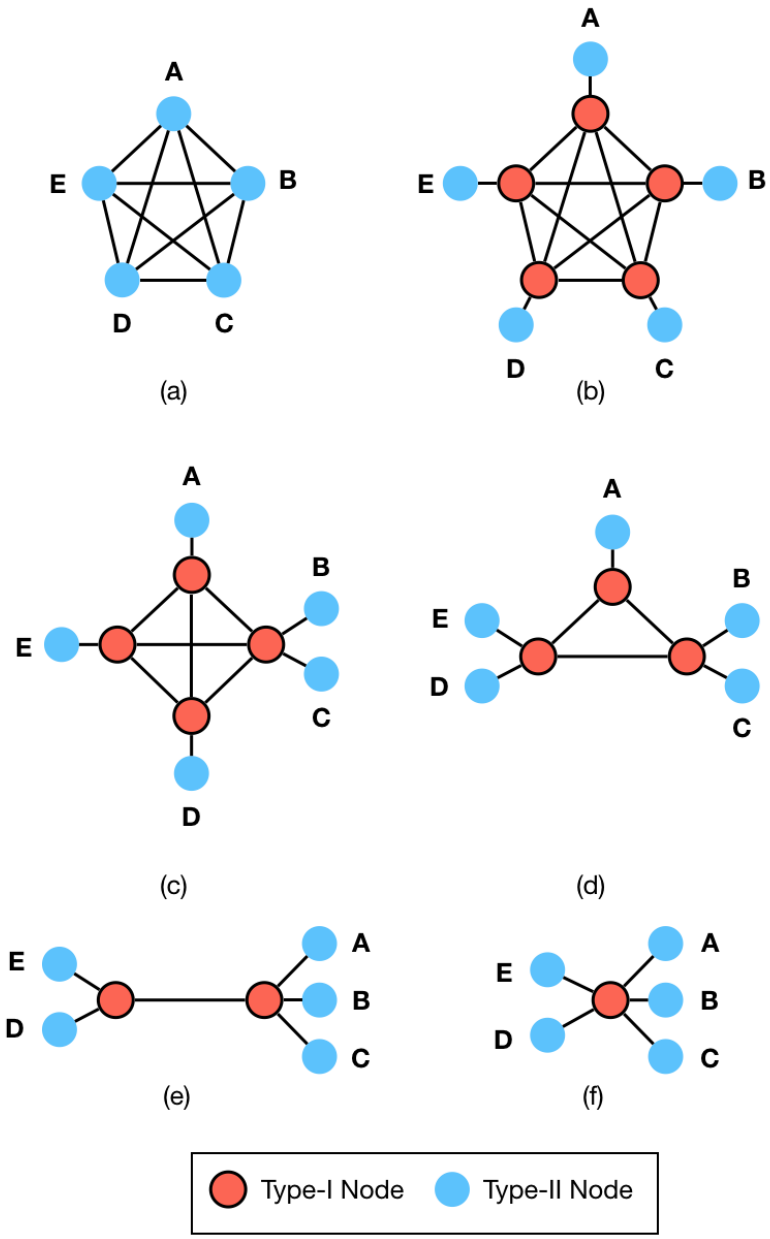


Figure 3.1: (a) The input graph is converted to such that (b) Type-II nodes are introduced for all components with an edge degree larger than 2 and connected to the original Type-I nodes. (c-f) Pairs of Type-I nodes that are connected through an edges are iteratively combined until no more pairs exist. In the case of the K_5 this results in a single Type-I node that will be replaced with a five way switch.

and $K_{3,3}$ (illustrated in Figure 2.4) as proven in Kuratowski's theorem [31]. If a non-planar graph is given as input for Directed Placement, then the planarization method introduced by Tseng et al. [59] can be used to pre-process the non-planar input into a planar one for placement, routing, and fabrication. This technique was not introduced in Chapter 2, as it was developed after those techniques were introduced, however it is non the less compatible with all forms of Planar Placement. A short description of this method follows here for completeness.

First, a new graph of the system is constructed with two different node types. The first is a Type-I node, which represents a switch that will be inserted into the system and can have an unconstrained number of edges. The second is a Type-II node, which represents any component within the system and will be constrained to having a maximum of two edges. The original input architecture is then processed with Type-II nodes representing each component, and if a given component has more than two edges a Type-I node is introduced with all the components original edges routing to the new Type-I node along with an additional edge between the Type-I node and the Type-II node representing the component. After the entire input has been processed in this way, the resulting graph is then iteratively reduced by combining any two Type-I nodes that connect through an edge. When all possible reductions of this type have been completed, then every Type-I node left in the system is replaced with a switch component capable of handling the number of edges associated with that node and the input graph has been planarized, and each Type-II node is replaced with the component it represents. A short example showing the planarization process of a K_5 subgraph can be seen in Section 3.3.1.

It should be noted that this method requires the insertion of switches into the system which require valves to operate, and can therefore only be used on active devices. Passive devices which are non-planar cannot be fabricated onto a single layer. Because the majority of microfluidic devices are being developed by hand with a single flow layer, they are naturally planar in nature. Should non-planar mVLSI netlists become prevalent (e.g., due to widespread adoption of mVLSI architecture synthesis tools [45, 39], none of which guarantee planar netlists), then this planarization technique will allow for a bridge between non-planar netlists and the current generation of placement and routing algorithms which require a planar input. While this technique does not guarantee that a small enough number of switches will be introduced to be feasibly fabricated, it does guarantee that any netlist can be minimally placed using these methods.

3.3.2 Initial Lane Assignment

As an optional first step, all input components $c_i \in I$ are added to the first lane L_0. Many microfluidic devices naturally place all of the inputs on one side, and, without loss of generality, during device operation, the fluid tends to flow from left to right. If I is not specified, the first step is to add the component $c_j \in C$ with the smallest $|T_j|$ to L_0, in the case of ties the component with the fewest ports $|P_j|$ is chosen, if there is still a tie choose randomly from the candidates.

A queue Q is created to facilitate a breadth-first traversal of the components. Initially, all components $c_j \in L_0$ are enqueued. The initial lane assignment heuristic proceeds until Q is empty.

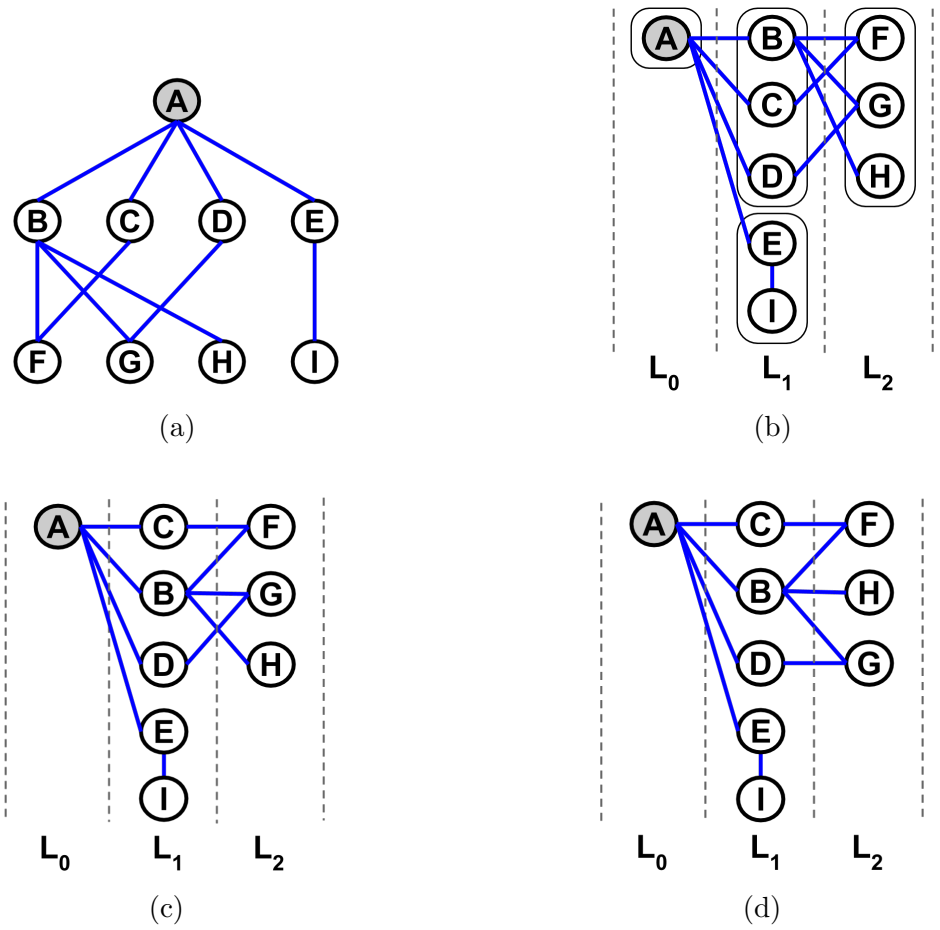


Figure 3.2: (a) The mVLSI input netlist is represented as an abstract graph, with components as nodes and connections as edges. In this example A is the only input. (b) Using a breadth-first traversal the nodes are assigned to an initial lane based on their traversal depth. Here the different subgroups are circled for illustration. Note that I is a terminal component so it is added to the same lane as its parent E. (c) Node B is moved to the center since it's subtree $\{F, G, H\}$ is the largest. (d) In L_2, nodes F and H are added first because they are processed from their last parent in the previous lane B. G is then added because it's last parent is D, which leads to a swap of G and H. This provides an abstract lane ordering but does not represent an actual placement

The first step is to dequeue a new component, c_q. Each neighbor $c_r \in T_q$ that has not yet been assigned to a lane is enqueued; c_r is also assigned to lane L_{f+1} where L_f is the lane to which c_q is assigned.

If c_r is a terminal component, then it is added to L_f to allow for a short connection (c_q, c_r); we enforce the constraint that both components are placed adjacent to one another within the lane. In order to minimize the lane width and simplify the later routing, a maximum of two terminal components connected to c_q may be placed in lane L_f and all additional terminal components connected to c_q are added to lane L_{f+1}.

If an mVLSI netlist consists of multiple connected components, then some components will not be assigned to a lane once Q is empty. This is unlikely to occur when placing and routing a single microfluidic device but may occur when performing these steps for a number of different devices on a single mask in order to increase production yields for mass manufacturing. If this occurs, the unassigned component c_j with the smallest degree $|T_j|$ is inserted into Q and initial lane assignment proceeds as normal. The process terminates when all components have been assigned a lane.

Figure 3.2a depicts an mVLSI netlist, and Figure 3.2b shows the initial lane assignment after the breadth-first search completes. In Figure 3.2b, components are grouped into subsets, as will be discussed in the next section.

3.3.3 Lane Ordering Optimization

Once each component has been assigned to a lane, those components need to be ordered within the lane to reduce route lengths. This is done by segmenting the components within a lane L_i into some number of ordered subsets $L_{i,0}$, $L_{i,1}$, ..., L_{i,m_i} such that now the lane L_i is an ordered set of ordered component subsets, the union of which contains all the components in the original lane $L_i = L_{i,0} \cup L_{i,1} \cup ... \cup L_{i,m_i}$. These ordered subsets continue to form a vertical arrangement of components, with the subset $L_{i,0}$ being at the top of

the lane and the subset L_{i,m_i} being at the bottom. Within these ordered subsets the first component $c_0 \in L_{i,0}$ will be placed at the top and the last component $c_{|L_{i,0}|-1} \in L_{i,0}$ will be placed at the bottom before the next subset $L_{i,1}$ begins to be placed within the lane. There are three stages to ordering the components within their lane:

1. **Subset Assignment:** Components within a lane L_i are assigned to a subset $L_{i,j}$ based on their parents in the preceding lane

2. **Subtree Ordering:** Components within a lane subset $L_{i,j}$ are ordered based on their subtree in successive lanes

3. **Parental Reordering:** Components within a lane subset $L_{i,j}$ are re-ordered based on the position of their parent components in the previous lane

Each step processes all components in all lanes before the next step begins.

Subset Assignment

In the first step, each lane L_i starting with L_0 is partitioned into subsets $L_{i,0}, L_{i,1}, ..., L_{i,m_i}$. In the first lane, L_0, each component $c_j \in L_i$ is partitioned into its own subset such that $m = |L_0|$. For each subsequent lane $L_i, i > 0$, the components $c_j \in L_i$ are partitioned into subsets based on their connections to components in the previous lane L_{i-1}. All $c_j \in L_i$ connected to the same $c_k \in L_{i-1}$ are partitioned into the same subset $L_{i,s}$, where s is the lowest unused subset index in the lane L_i. If c_j connects to multiple components c_k in L_{i-1}, it is partitioned into the first possible subset. Figure 3.2b depicts the components in three lanes partitioned into subsets.

Subtree Ordering

The second step begins after all components $c_j \in C$ have been partitioned into some subset $L_{i,s}$. During this step, all lanes L_i and subsets within lanes $L_{i,j}$ are traversed via indices $0 \leq i \leq |L| - 1$ and $0 \leq j \leq m_i$; recall that m_i is the number of subsets in lane L_i.

First, each component $c_k \in L_{i,j}$ is sorted based on the length of its subtree in subsequent lanes $L_p, p > i$. The subtree length is determined using a breadth-first traversal starting from c_k. If the search is presently processing component c_j in lane L_b, then it is not allowed to expand to any components belonging to lane L_a where $a < b$. The number of components traversed is then used to sort the components within the lane subset, with the component with the largest subtree in $L_{i,j}$ being at the center of the subset and subsequent components being ordered away from the center. This is illustrated in Figure 3.2c.

Parental Reordering

Once the components have been ordered based on their subtree size, the third step is to re-order them to remove edge crossings between lanes. When the components with large subtrees are moved toward the center in the previous step, doing so can increase the number of intersections between lanes. Parent reordering tries to re-order the components based on their parents' locations to remove these intersections. A new lane L_t is created to temporarily store the new ordering of the components during the re-ordering. The lanes $L_{i,j}$ are iterated in reverse order from $i = |L| - 1$ to 1 and component in forward order $j = 0$ to m_i. For each component $c_k \in L_{i,j}$ from the top of the subset to the bottom, the algorithm searches through c_k's neighbors in the previous lane L_{i-1} and adds them to L_t based on

their ordering in L_{i-1}. If a component in L_{i-1} is a neighbor of multiple components in L_i, then it is added to L_t when processing its last neighbor in L_i. Any components in L_{i-1} not connected to a component in L_i are then added to L_t, and the previous lane L_{i-1} is updated to $L_{i-1} = L_t$. This is illustrated in Figure 3.2d.

The same steps are performed in the opposite direction, iterating the lanes from $i = 0$ to $|L| - 2$, and $j = 0$ to m_i. In this iteration, for all components $c_k \in L_{i,j}$ from the top of the subset to the bottom we will identify neighbors in the next lane L_{i+1} and add these components to L_t, with the rest of the process continuing as previously described, and updating L_{i+1} to the ordering of L_t.

3.3.4 Component Rotation & Port Assignment

The previous ordering steps mean that components are in optimized locations relative to their neighbors, but it does not mean that the ports of those components are located in a good position for routing. This necessitates a component rotation step before components can be given a location and routing can be performed.

The source and sink of a connection in input architecture can be either *port assigned* or *port unassigned*. When a connection's source and/or sink is port assigned, then it is required to route to a specific port on the component it is connected to. This occurs because the component that it routes to is functionally dependent on fluids flowing into its input ports and out of its output ports. An example of this would be a rotary mixer, which requires fluids to flow in through a certain port in order for the valve actuation sequence to correctly input and mix the two fluids. When a connection's source and/or sink is port

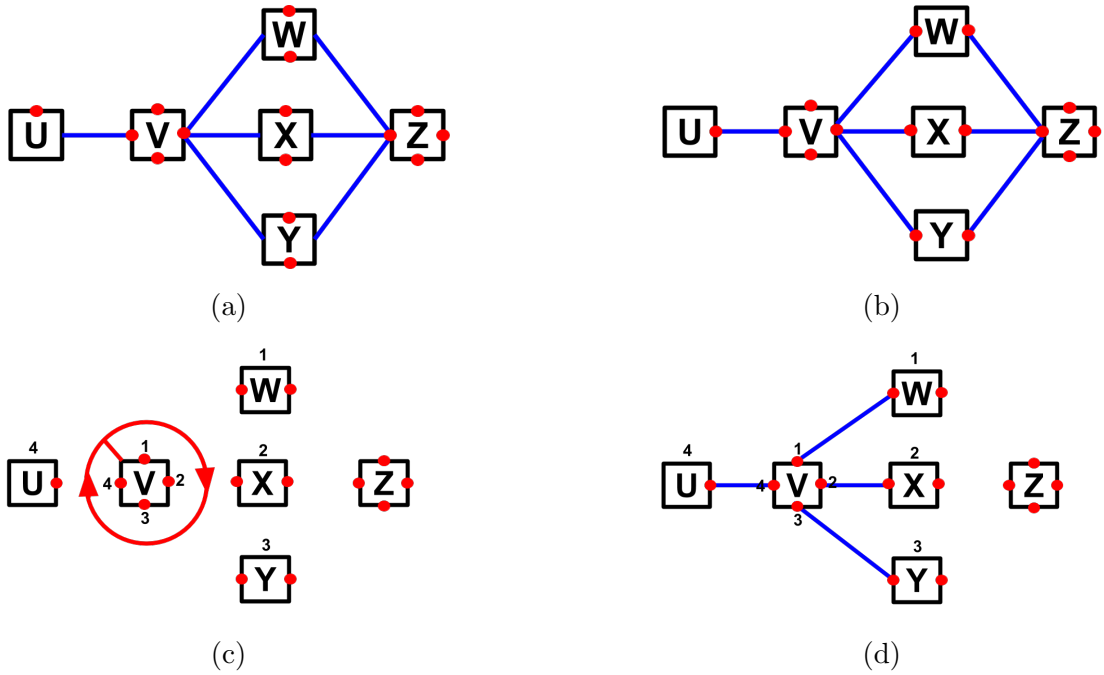


Figure 3.3: (a) The mVLSI device after placement. (b) The components are rotated based on a weight function so that the number of ports facing connected components is maximized, causing U, W, X, and Y to rotate. (c-d) Port assignment is performed on component V, which performs a radar sweep to determine processing order. In this case, the Manhattan distance used for port assignment matches the radar sweep ordering.

unassigned, it does not have a specific port on its source/sink component that it needs to route to and can be routed to any port that is not already port assigned. This usually occurs on components that can function in any direction equally well, such as cameras and detection mechanisms.

In order to account for this, for each component $c_i \in C$ a weight is calculated for the component with rotations of 0 90 180 and 270 which are the only orientations that are allowed because of the grid based routing that is performed. For each orientation the weight is calculated to be the sum of the number of port assigned connections with their matching connected component in that same direction and the sum of the lesser of the number of

unassigned ports or the number of connected port unassigned components. This value is calculated for each side of the component and its corresponding direction.

That is, for a component located at $L_{i,j}$ the weight in the east direction would be the number of assigned ports on the east side of the components who's connected components exist in a lane east of the component $(L_k, k > i)$ summed with the lesser of the number of unassigned ports on the east side of the component or the number of port unassigned connected components in a lane east of the component $(L_k, k > i)$. This is then calculated for the ports on the west side $(L_k, k < i)$, the north side $(L_{i,k}, k < j)$, and the south side $(L_{i,k}, k > j)$. These values are then summed to create the weight for that particular component orientation. The weight for each orientation is then calculated, and the orientation with the highest weight chosen.

Once the component has been rotated, each port unassigned source and/or sink on each connection must be assigned. For each component $c_u \in C$ with an unassigned port from L_0 to $L_{|L|-1}$, processing from the top of the lane to the bottom, we perform a radar sweep similar to the one described in [7] beginning in the upper left corner of the component. As the radar sweep passes components, if it sweeps past a component $c_v \in T_u$ then the associated edge $e_z = (c_u, c_v) \in E$ is processed. The Manhattan distance between each unassigned port in the source component $p_j \in P_u$ and each unassigned port in the sink component $p_k \in P_v$. The combination of ports with the minimum Manhattan distance is then assigned as the source and sink ports for that connection, and p_j, p_k are no longer candidate ports for later assignments. This process continues until all connections with unassigned ports have been assigned.

3.3.5 In-lane Placement

At this point all components have been assigned to a lane, have been given an order within each lane, and have been rotated to optimize connection routing. However, the components have not yet been assigned an (x, y) coordinate for placement. An initial y-coordinate can be determined for each component by iterating through each lane L_i and placing the components in-order, with appropriate spacing between them. The first component $c_0 \in L_i$ is given a y-coordinate of $y_0 = \Delta$ (assuming the top left of our 2D plane is the original at $(0, 0)$). This ensure that all components have enough distance from the edge of the device for routing and fabrication. Each subsequent component $c_j \in L_i$ is then placed at the position $y_j = y_{j-1} + h_{j-1} + \Delta$, which is the y-coordinate of the previous component placed shifted to account for the height of the component and an adjustable spacing quanta, Δ.

From here, the components are adjusted to better align with their parents' in the preceding lane. The purpose is to improve routability and to try to create routes between lanes that are of similar length. For each component $c_j \in L_i, i > 0$ a new set of components $V = \{c_k \in L_{i-1} | (c_j, c_k) \in E\}$ is created. If $|V| > 1$ then the component c_j is shifted to align with the average y-coordinate of the parent components in V. A shift factor (δ) is calculated, such that:

$$\delta = \frac{\sum_{i=1}^{|V|} y_{V[i]} + (h_{V[i]}/2)}{|V|} - y_j$$

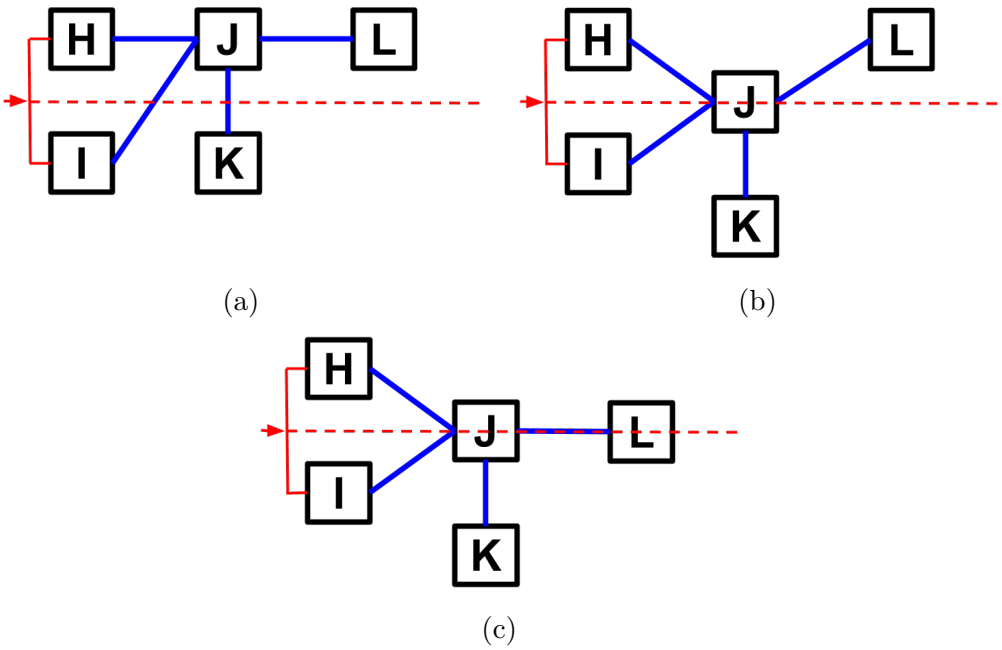
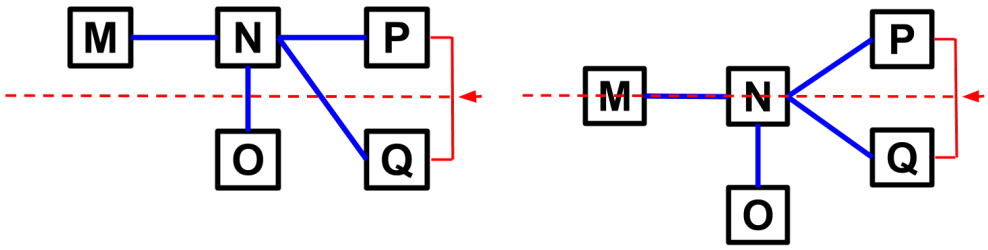


Figure 3.4: (a) The horizontal red line represents the vertical center of the parents of component J, calculated as the average of the y-coordinate of each parent component $\{H, I\}$. (b) Component J is shifted to the center of its parents, shifting the other components in the lane, K, by the same amount. (c) All other components in subsequent lanes, L in this case, must also be shifted down by that amount.

In the case where $|V| = 1$, V is redefined as $V = \{c_j \in L_i | (c_k, c_j) \in E\}$, and all the components in V are shifted such that the average y-coordinate of all the components in V align with the center of component c_k. In this case, the shift factor is calculated such that:

$$\delta = \frac{\sum_{i=1}^{|V|} y_{V[i]} + (h_{V[i]}/2)}{|V|} - y_k$$

If either c_j is shifted or the set of components in V are shifted, additional components in the lane L_i must be shifted to avoid intersections. Shifting a component c_j (set of components V) requires the movement of all the components in L_a, where $a < i$, and



(a) (b)

Figure 3.5: (a) During a backward iteration when processing component P, the component has only a single parent N. This causes the subtree of the parent component N, which contains $\{P, Q\}$ to be shifted instead of P itself. (b) The subset $\{P, Q\}$ is shifted down to the center of their parent N. Since there are no other components in that lane and no subsequent lanes, no other components need to be shifted.

the rest of L_i to prevent overlap. If $\delta < 0$, we shift c_j (all components in V) upwards and need to ensure no components are moved above the chip's boundaries. That is, we must maintain for each $c_t \in C$, $x_t \geq \Delta$ and $y_t \geq \Delta$. We first shift all $c_t \in C$ downwards by $|\delta|$. So for each $c_t \in C$, $y_t = y_t + |\delta|$.

Finally, shift the remaining elements of L_a by δ. For each $c_t \in L_a$ where $y_t > y_j$, c_t is moved such that $y_t = y_t + \delta$. Any terminal components connected to a component c_t should also be shifted by δ. Components in L_a with $a < i$ are shifted by δ as well. At this point the set V is emptied and the process continues with the next component. Figure 3.4 illustrate the shifting of the single component and subsequent components while Figure 3.5 illustrates the shifting of the component set.

If additional padding is required around the edge of the device to meet fabrication requirements, it can now be added. The entire device can be shifted and/or the size of the device can be increased to accommodate any padding requirements.

3.3.6 In-lane Horizontal Centering

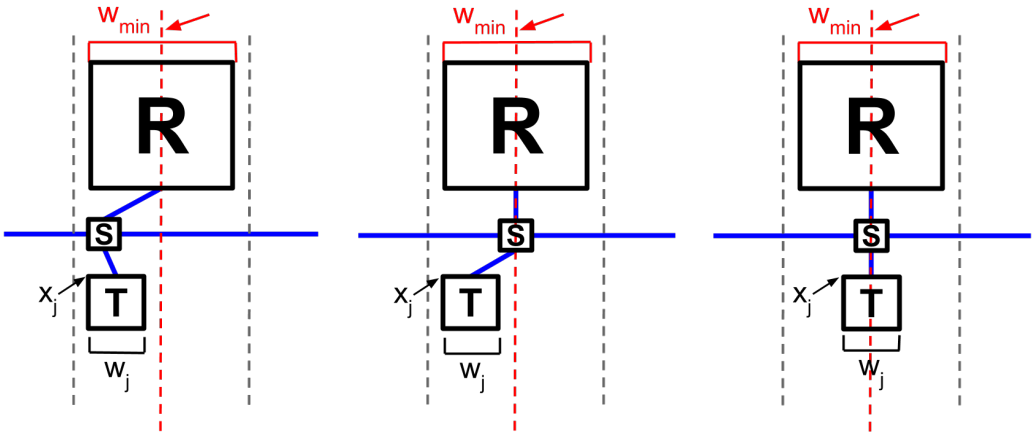


Figure 3.6: (a) The vertical red lines show the calculated horizontal center of the lane based on the widest component, R in this case. (b-c) The center of component S and T are shifted to the lane center.

The next step is to determine each component's x-coordinate. This process begins by iterating through each lane L_i from $i = 0$ to $|L|-1$. For the first lane L_0, all components are given an x-coordinate equal to the buffer space, $x_j = \Delta, \forall c_j \in L_0$. This ensures that all components in the left most lane have enough distance from the edge of the device for routing and fabrication.

Next, the minimum width of the lane (w_{min}) is found. To prevent overlapping components between the lanes, the minimum width of the lane is equal to the component with the largest width such that $w_{min} = min(w_j), \forall c_j \in L_i$.

Once w_{min} is determined, a second iteration of all components $c_j \in L_i$ is made to center each component within the lane. Each component's x/coordinate is shifted to

center the component within it's lane based on the following equation, which is illustrated in Figure 3.6

$$x_j = x_j + \frac{w_{min} - w_j}{2}$$

Once all the components in the lane L_i have had their x-coordinate re-centered within the lane, the lane iteration continues. For all lanes $L_i, i > 0$, instead of setting all components $c_j \in L_i$ initial x-coordinate $x_j = \Delta$ the initial x-coordinate is set to $x_j = x_0 \in L_{i-1} + w_{min} + \Delta$. This ensures that all the components in the next lane are far enough to the right of the previous lane to ensure there is no component overlap between lanes with the additional buffer space needed to improve routability and meet fabrication requirements.

3.4 Routing

3.4.1 Flow Layer Routing

Once the components have been placed and all connections assigned ports, the routing of the connections is performed using a slight modification to the method described in Ref. [37]. A brief description of that method as well as the modifications to it is provided for completeness. A routing grid $R = (U, F)$, where U is a set of grid points, and F is a set of edges representing potential channel routes between adjacent grid points. For each component $c_i \in C$ a vertex u_i for the ports $p_h \in P_i$ is instantiated and added to U. A grid of vertices is then instantiated in the empty space between components. Edges that represent potential routing channel segments are added to F by instantiating a bidirectional

edge f_i with a capacity of 1 between $u_i \in U$ and $u_j \in U$ if and only if $(u_j.x - u_i.x == 1) \oplus (u_j.y - u_i.y == 1)$.

Once the grid $R = (U, F)$ has been constructed, the next step is to route channels between the components. Unlike in Ref. [37], where a network-flow based router is utilized to do routing and port assignment, port assignment has already been completed. Instead of a network-flow based routing; for each port $p_j \in P_i$ of component c_i that has a connection assigned to it, a breadth-first search is made start from the source port p_j until it reaches that connections assigned sink port p_k. A path reclamation step adapted from Lee's algorithm [33] is then performed to find the shortest path from the sink p_k to the source p_j. The reclaimed path is then set as the final route for that connection and its grid point are marked as unusable for future routes. If there is a minimum padding between connections required for fabrication reasons, then that number of additional units away from the route are also marked as unusable. This process is repeated for every connection in the system.

3.4.2 Control Layer Considerations

While routing of the control layer is out of scope, the method presented here does allow for placement relaxation that can be useful when routing the control layer. Since Directed Placement places flow-layer components in a left to right orientation, it is advised that control layer I/O should be placed along the top and/or bottom edge of the device. From here, control lines can be routed through the buffer space between lanes or directly through components (where fabrication allows) to the edges. It is also possible for the inter-lane buffer space to be utilized by pin insertion methods [25] to insert control pins closer to the components that they control and reduce control route length.

In both these cases, the amount of unused space that can be utilized for control routing can be increased in a targeted manner through the manipulation of the lane buffer space Δ for a subset of lanes. If, for example, a component $c_j \in L_i$ was unable to be routed to a viable control pin, then the Δ between lanes L_{i-1}, L_i and L_i, L_{i+1} could be increased by some value σ to allow more space for pin insertion or control line routing. This increase of σ would then re-trigger the in-lane placement and routing steps, and another attempt by the control routing method to find a set of valid routes. This process could be performed iteratively unless a valid control routing was found, or a maximum size threshold was reached.

3.5 Results

The Directed Placement paired with the Lees' based router described here is compared to the Planar Placement Baseline Expansion (BaseEx) and Diagonal Component Expansion (DICE) methods paired with the Network-flow based routing algorithm as described in Chapter 2 and the Simulated Annealing (SA) based placer [41] paired with Hadlock's maze routing algorithm [45] also introduced in Chapter 2. All of these algorithms were implemented in C++ utilizing a a *unitless* grid, which decouples the layout and design rule checking processes from the manufacturing resolution of any one specific mVLSI technology [42].

3.5.1 Benchmarks

We obtained netlists for four real-world planar mVLSI devices that have been designed, fabricated, and evaluated, as well as five netlists obtained by synthesizing synthetic benchmarks. The real-world netlists are as follows: aquaflex-3b & aquaflex-5a are proprietary mVLSI netlists provided by Microfluidic Innovations, LLC, hiv is a multi-layer PDMS chip that performs a bead-based HIV1 p24 sandwich immunoassay [34] and mgg is a molecular gradients generator that can generate five concentration levels of a two-sample mixture [52]. The five synthetic benchmarks were generated by compiling a set of publicly available DAG specifications [22] through an established mVLSI architectural synthesis flow [45, 39]. Additional information on these benchmarks as well as the number of connections, components, and average component area of each benchmark can be found in Section 6.3.2 and Table 6.2.

3.5.2 Results and Analysis

For each benchmark, we report the area utilization (Figure 3.7: the ratio of component area to total chip area expressed as a percentage), average fluid channel length (Figure 3.8a), average fluid channel length reduction (Figure 3.8b), and average runtime (Figure 3.9). Directed Placement and Planar Placement (both BaseEx and DICE) achieved planar layouts in all cases while SA did not. SA was not designed to minimize the number of intersections during placement or routing, and is highly unlikely to produce a routing with no intersections. We do not report the number of crossings in the layouts produced by SA, but the number was nonzero in all cases.

Unlike in Chapter 2 we modify the SA method used here to yield better results. We first remove the component segmentation requirement from SA which caused it to be tied very closed to its initial placement. Additionally, we implement a sweeping mechanism for finding the minimum size that could be used to still generate an initial placement. Finally, we still allow for the Hadlocks-based router to introduce an infinite number of intersections, creating unrealistic layouts. These modifications make the SA method presented here a proxy for a closer-to-optimal layout.

Area Utilization

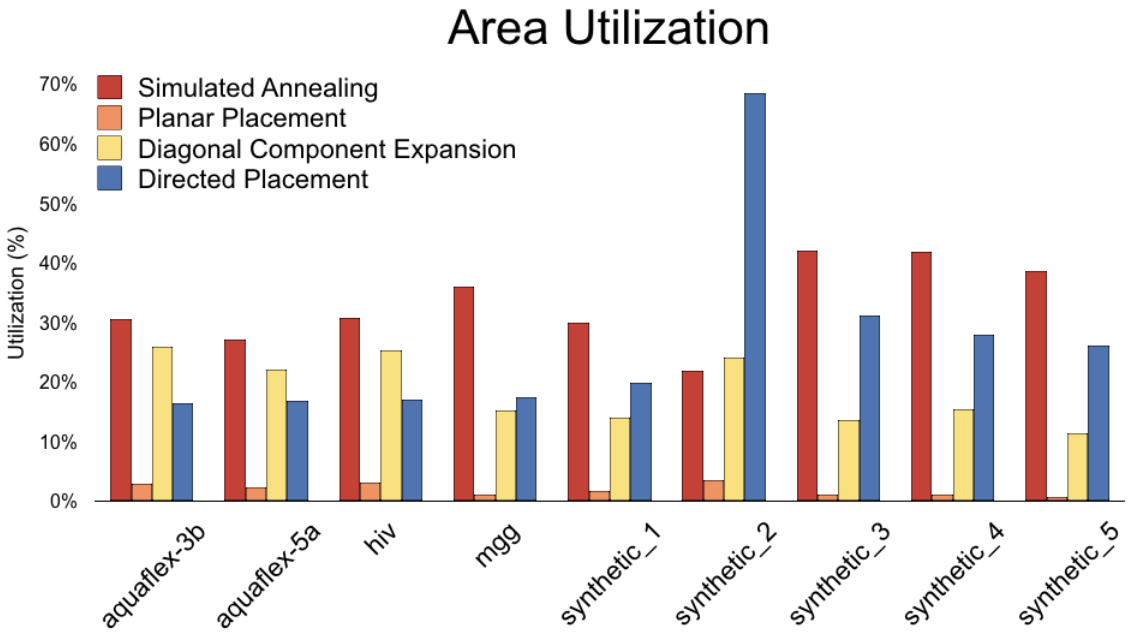


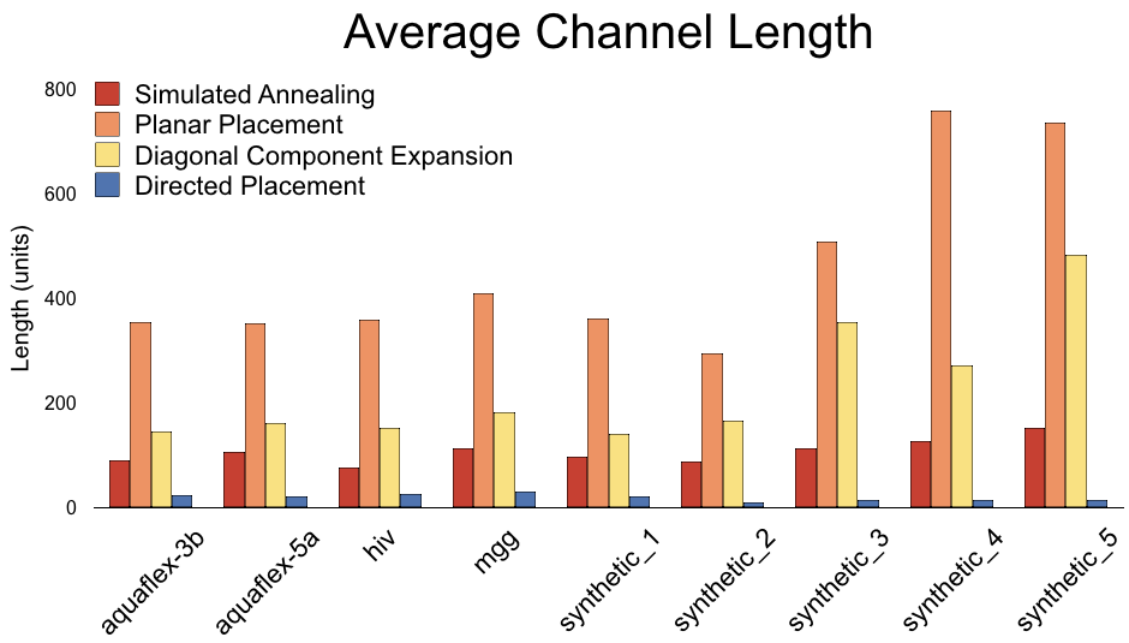
Figure 3.7: The sum of the area of all the components in the device divided by the total area required to place and route the device, represented as a percentage per benchmark.

In Figure 3.7, SA achieves the highest area utilization for all the test cases except one. This result is expected since the SA method is primarily focused on optimizing the

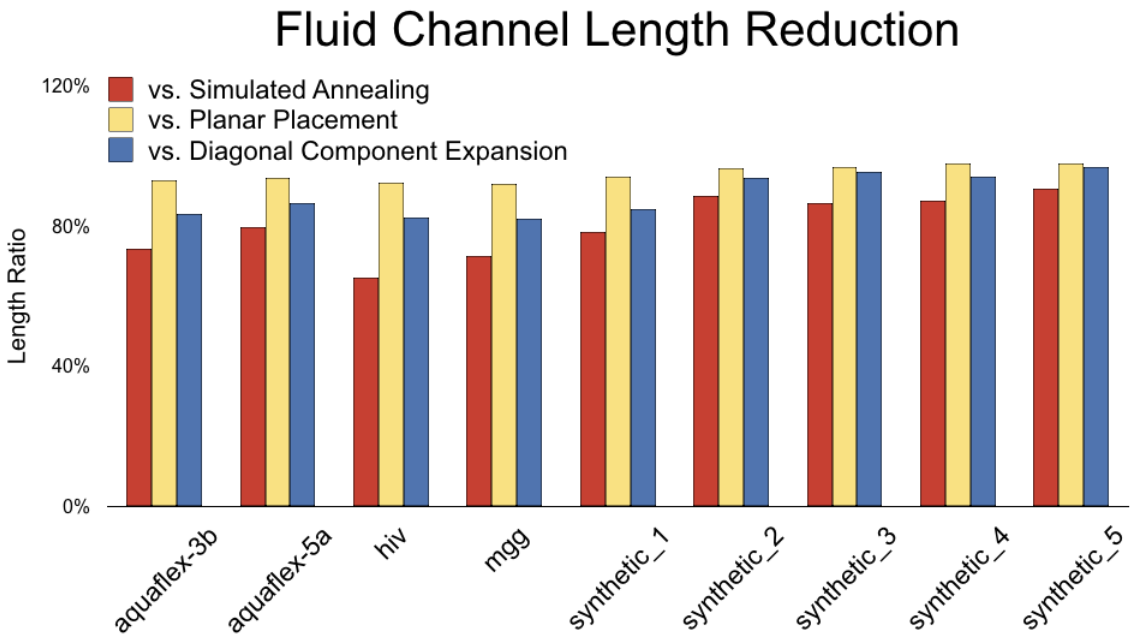
total area of the device and ignores the requirement that no routes intersect in the system. The one benchmark that SA is not best suited for is synthetic_2. Directed Placement and DICE are especially effective on the synthetic_2 benchmark, increasing its area utilization from 22.65% with SA, 3.60% with BaseEx, and 24.2% with DICE to 68.57% with Directed Placement. This dramatic increase on this particular benchmark is due its particularly linear nature, yielding a straight line layout with Directed Placement and a relatively linear placement in DICE. The rest of the benchmarks have a more complex architecture and do not allow for this type of straight line placement. On average Directed Placement is 81.60% as effective as the SA method for area utilization.

Fluid Channel Length

For all benchmarks in Figure 3.8a, Directed Placement achieves the shortest average fluid channel length. This is because Directed Placement utilizes the tree-like structure of mVLSI devices to create designs that place neighboring components as close as possible. SA, in contrast, starts with a random placement and seeks to primarily optimize the total area and intersections. BaseEx utilizes a planar embedding for its initial placement, however these planar embedding algorithms tend to lay out the components into triangular substructures with increasing straight line distances between them. This leads to small densely packed subgroups with large distances between them. DICE arranges the components across the diagonal of the layout, which helps to increase area utilization but often yields many routes that must cross a large portion of the layout in order to connect. Additionally, the Network-flow based routing algorithm used with both BaseEx and DICE, like the one employed



(a) The average length of all the fluid channels present in the device per benchmark.



(b) The percent reduction in the average fluid channel route length when compared against Directed Placement per benchmark.

with Directed Placement, actively avoids introducing intersections into the system. SA is not bounded by this requirement which allows for the second shortest routes across all

connections. As shown in Figure 3.8b, Directed Placement reduces fluid channel length in all cases while also avoiding adding any additional intersections to any of the benchmarks.

Runtime

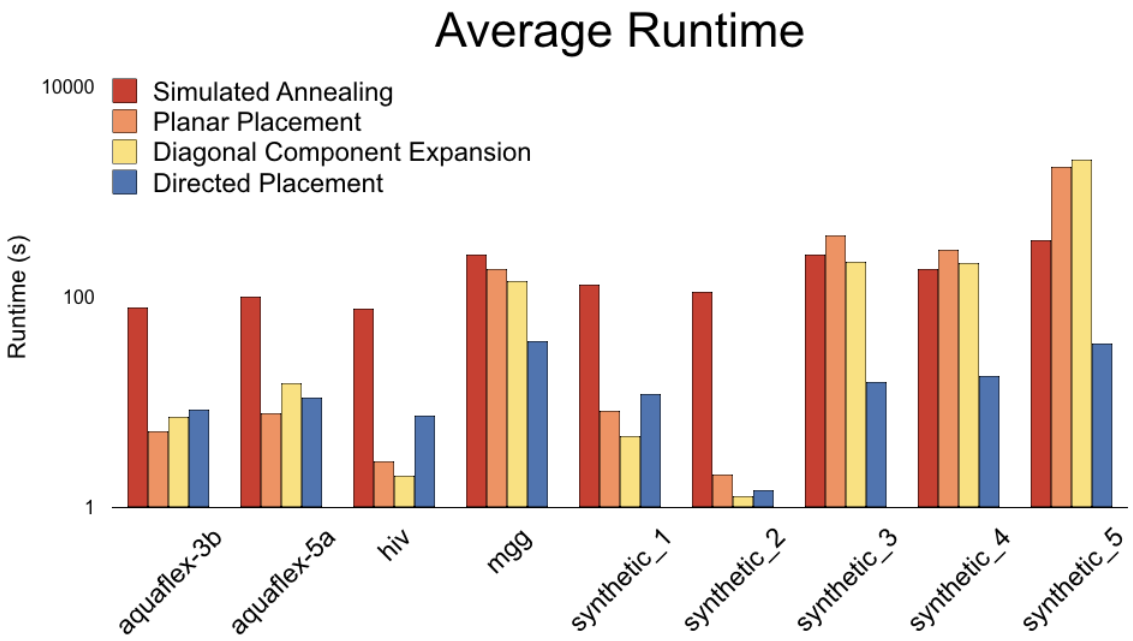


Figure 3.9: The average runtime of all algorithms over five runs per benchmark.

Figure 3.9 shows the average runtime of each algorithm for each benchmark over five runs. SA has variable parameters that will effect both its runtime and the solution that it converges to. For the results presented here SA was run with $100,000$ moves per temperature change, a cooling rate of 1%, and an initial temperature of 100. When Directed Placement is compared to BaseEx and DICE, BaseEx and DICE tend to run faster on smaller benchmarks while Directed Placement runs faster on larger ones. This occurs because the Directed Placement algorithm is more complex than BaseEx and DICE, but yields a better placement for the routing step. Since all three methods utilize the same or similar routing steps, on

small test cases where the routing makes up a small portion of the runtime BaseEx and DICE run faster but as the routing requirements increase Directed Placements superior layout means a shorter routing time and a faster overall runtime. The one exception to this is the synthetic_2 benchmark, which runs fastest on Diagonal Component Expansion while still yielding a longer average fluid channel length. This is because the straight line nature of that particular benchmark are trivial for the expansion method used in DICE and yields long fluid channels with few possibilities for intersections. Since the Network-flow based router will perform a rip and re-route step if an intersection occurs, a reduction in possible intersections leads to a large reduction in the overall runtime. Because the SA method uses a Hadlocks-based router that does not avoid intersections, the vast majority of the time reported is spent in the placement stage. All other methods spend the majority of their time performing the routing step.

3.6 Conclusion

Directed Placement algorithmically mimics one of the typical design layout strategies employed by real-world microfluidic designers. As a result, it generates designs that are efficient, with relatively high area utilization and short channel lengths; however, it is designed primarily for microfluidic netlists that implement assays that are specified as a relatively linear sequence of steps. Although this may seem limiting, linear designs represent a large portion of microfluidic devices currently under development today. On the other hand, Directed Placement produces relatively poor layouts for grid-based netlists or devices that feature internal loops. In principle, there is room to develop specialized layout algorithms

for netlists that fall into these categories as well, although we do not attempt to do so here; Chapter 6 contains several initial steps toward this objective.

Chapter 4

Seam Carving-based Post Processing

4.1 Introduction

End-to-end microfluidic design automation starting from a high-level language specification of a biochemical reaction would be ideal; however, real-world microfluidic device designers today do not yet trust this level of automation and would be reticent to use any tool that removes their ability to manually intervene at each step of the design process [40]. New microfluidic components in particular are highly dependent on specialized device geometries and must be designed manually, often in conjunction with fluid modeling software. Microfluidic device designers recognize the need for component libraries akin to standard cells [1], but strongly prefer graphical design interfaces over end-to-end algorithmic solutions.

[1] This sentiment was clearly expressed by Emmanuel Delamarche (IBM, Zurich) in a special session talk at DATE 2018

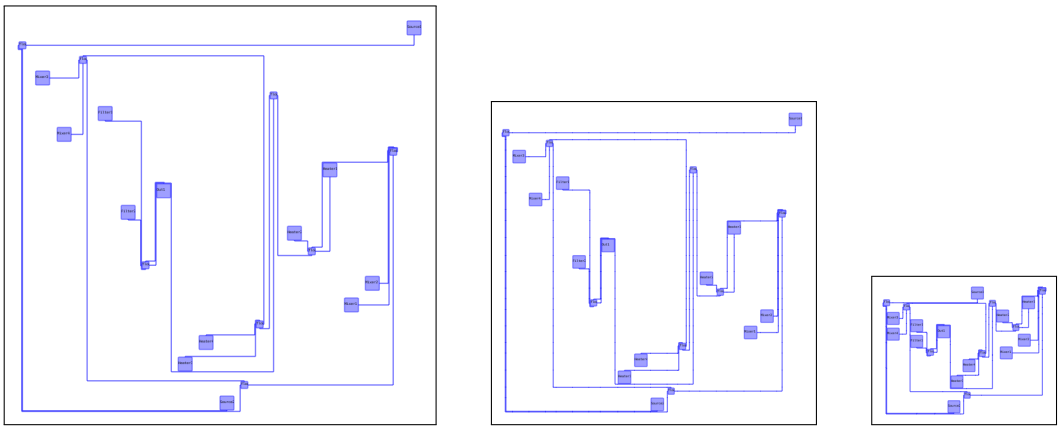
In order to be able to support designers' current methods as well as algorithmic advances, we developed a number of post-processing techniques that are agnostic to the method used to generate the layout. These algorithms can process layouts generated by hand as well as those that are algorithmically generated. The first method we introduce is one that adapts seam carving [4], an image size reduction technique, to reduce the device area and channel lengths of sub-optimal microfluidic very-large-scale integration (mVLSI) layouts. Currently, the only constraint to this technique is that channels must be routed rectilinearly.

The basic premise of the technique we describe here is to identify seams (paths) through the chip which can be removed without adversely affecting device functionality, shortening fluid channels and reducing the devices overall size. Figure 4.1 shows a motivating example. Figure 4.1a shows a low quality initial layout [38] created using the Planar Placement and Network-flow routing technique from Chapter 2. Figure 4.1b shows an improved layout, which was derived using linear seam carving, which we introduce in Section 4.4. Finally, Figure 4.1c shows a better result which was obtained with a more aggressive technique, nonlinear seam carving, which we introduce in Section 4.5.

4.2 Related Work

4.2.1 Seam Carving

A seam is a path of pixels through an image whose removal minimally degrades image quality. Seams can be identified by converting an image into a weighted graph, where each



(a) (b) (c)

Figure 4.1: (a) Shows the benchmarks synthetic_1 after the baseline placement and routing [38] has completed. (b) is the same benchmark after linear seam carving has been applied, while (c) is after non-linear seam carving has been applied.

vertex represents a pixel and each vertex's weight represents its relative importance to image quality [4]. Seam carving then finds the lowest-cost path from one perimeter edge to its opposite and removes it from the image. The process repeats until the desired reduction in size is achieved.

Existing seam carving techniques cannot directly be applied to mVLSI devices. Components are usually designed to have specific geometries that cannot be modified without adversely affecting their functionality. Thus, the seam identification process must explicitly exclude them. This requirement is also incompatible with other image re-sizing techniques such as scaling [11], which expands or compresses an image by a scale factor.

4.2.2 mVLSI Placement

Seam carving can reduce the area of mVLSI chips designed manually, or laid out using sub-optimal heuristic methods such as Simulated Annealing (SA) [45], Incremental Cluster Expansion (ICE) [56] and extensions to planar graph embedding [38]. Seam carving will not be able to improve an optimal placement result [59] because the existence of a removable seam contradicts the optimality of the result.

We are aware of one mVLSI post-processing step, which repeatedly selects components on the perimeter of the layout and uses binary search to determine how far they can be moved toward the center [38]. In our experiments, we found that the runtime of the binary search method to be prohibitive, primarily because each potential movement rips up and reroutes fluid channels incident to the component that was moved. Seam carving, in contrast, does not recompute the placement of components or routing of fluid channels, and runs considerably more efficiently as a result.

4.3 Preliminaries

The input to seam carving is a placed and routed mVLSI architecture $A = (C, R, n, m)$, where C is a set of placed components, R is a set of routed channel segments, and n and m are the respective height and width of the layout. We represent each microfluidic component $c_i = (x_i, y_i, w_i, h_i)$ using a bounding box: point (x_i, y_i) is the upper-left corner of the component, and h_i and w_i are its respective height and width. Each routed channel segment $r_i = (x_{i,t}, y_{i,t}, x_{i,l}, y_{i,l})$ is a straight-line connection between points $(x_{i,t}, y_{i,t})$ and $(x_{i,l}, y_{i,l})$; multiple segments may comprise a longer channel with twists and bends. The

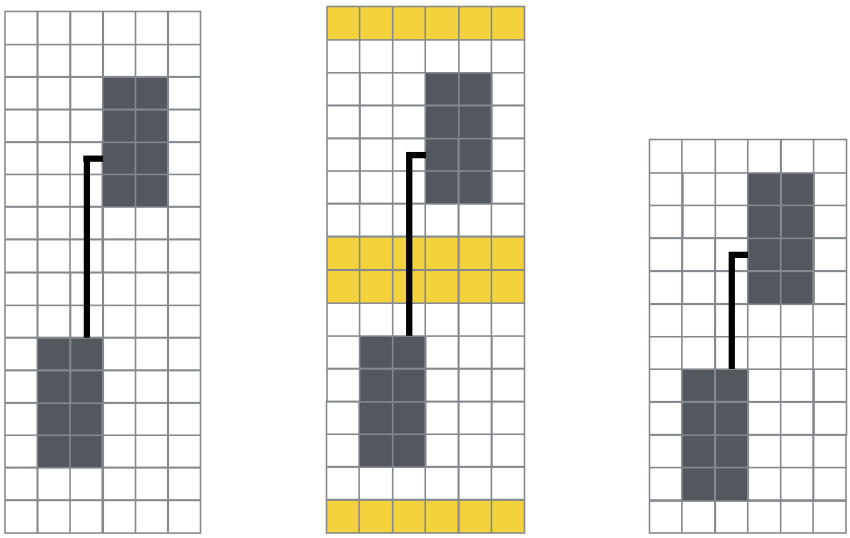
physical layout process may include an additional parameter, Δ, which adds white space around each component to improve routability.

In microfluidic devices all space in an architecture can be classified into three categories. Components, which have a fixed height and width; we assume that component dimensions are fixed by fluidic IP designers and cannot be reduced without adversely affecting chip functionality. Fluidic channels, which can be of any length as long as they provide a continuous flow of fluid between source and sink components; channel length can be reduced without altering chip functionality. Free space, which is superfluous, except for the buffer space surrounding each component.

A seam is a path through the architecture that connects one perimeter edge to its opposite and contains no points that are invalid for removal; this ensures that correct device functionality is maintained when the seam is removed. Invalid points include any part of a component (including its buffer space) or a switch at a channel intersection. In the latter case, removal of a switch would require the post-processor to re-place the switch and reroute its incident fluid channels accordingly; it is preferable to avoid this overhead. Valid points for removal include free space and channel segments that are not part of a switch and would not break the route connection between connected components.

4.4 Linear Seam Carving

Linear seam carving restricts seams to be horizontal or vertical straight lines that do not bend. Figure 4.2a shows an example mVLSI chip with a loose placement and ample white space. Figure 4.2b shows four horizontal seams, two of which intersect fluid channels in the



(a) (b) (c)

Figure 4.2: (a) A laid out mVLSI chip; (b) seam identification ($\Delta = 1$); (c) the chip after seam removal.

center of the chip. Figure 4.2c shows the smaller chip after the four seams are removed. Device functionality is not altered, and the channel connecting the two components is shortened but not disrupted.

4.4.1 Seam Identification

Linear seam carving employs two boolean arrays, B_x and B_y, which respectively represent removable vertical and horizontal seams. Without loss of generality, as we move along the x-axis, $B_x[i]$ represents a vertical line containing all points within the component having i as the x-coordinate. Both arrays are initialized to $B_x[0:m] = B_y[0:n] = True$.

The algorithm identifies vertical and horizontal seams for removal separately. To identify vertical seams, the algorithm iterates through all components $c_i \in C$ setting $B_x[x_i : x_i + w_i] = False$; this disallows any seam that cuts through a component. For each route

$r_i \in R$ the algorithm sets $B_x[x_{i,t}] = False$ and $B_x[x_{i,l}] = False$ to disallow the removal of switches at channel intersections. Any index i for which $B_x[i] = True$ represents a vertical seam that could be removed. Horizontal seams are identified similarly, using B_y and the y-coordinates of components and channel segments.

Seams are permitted to cut through channel segments, effectively shortening them. If a channel segment of a pre-specified length is required (e.g., to achieve a chemical separation), then it should be characterized as a component.

4.4.2 Seam Carving

Each index $j \in \{0, ..., m\}$ where $B_x[j]$ is $True$ is a removable vertical seam. Each component $c_i \in C$ such that $x_i > j$ is shifted left to fill the space removed by the seam; the height and width of c_i remain unchanged. Channel segments completely to the right of the removed seam are shifted left by one grid point. For channel segments that cross the seam, the right endpoint is shifted left by one grid point. Seam carving cannot completely remove a channel because seams cannot contain channel endpoints. The final step is to reduce the length of the guide B_x by one grid point by setting $B_x[k] = B_x[k+1], j \leq k \leq m$, and decrementing m. This process then repeats similarly for all vertical seams, $0 \leq j \leq n$ where $B_y[j]$ is $True$.

4.5 Non-linear Seam Carving

Non-linear seam carving eliminates the restriction that seams are exclusively horizontal or vertical segments. Seams are still required to begin at one perimeter edge and end at the opposite edge. This increases opportunities for seam removal and can lead to substantially smaller chip designs.

4.5.1 Seam Identification

Seam identification employs an $m \times n$ Boolean grid G to determine if a given point is a candidate for seam carving. All grid entries are initialized to $True$. For each component $c_i \in C$ at position (x_i, y_i) we set $G[j][k] = False$, $x_i \leq j \leq x_i + w_i, y_i \leq k \leq y_i + h_i$, rendering these points invalid for inclusion in a seam. For each routed channel segment $r_i \in R$ we set $G[x_{i,t}][y_{i,t}] = G[x_{i,l}][y_{i,l}] = False$ to disallow seam carving through switches at channel intersection points. A seam S is a collection of straight line segments $s_i = (a_i, b_i, c_i, d_i)$, where (a_i, b_i) and (c_i, d_i) are the (x, y)-coordinates of the two endpoints.

Non-linear seam carving retains the directional approach of its linear counterpart. Seams are first identified along the x-axis, with an artificial *source* connected to all grid positions $G[j][0], 0 \leq j \leq m$ and and artificial *sink* connected to all grid positions $G[j][n], 0 \leq j \leq m$, as shown in Figure 4.3a. Lee's Algorithm [33] is repeatedly called to identify seams from *source* to *sink*, until no valid paths remain. Figure 4.3b shows two non-linear seams, whose removal yields the smaller chip depicted in Figure 4.3c. This process then repeats along the y-axis.

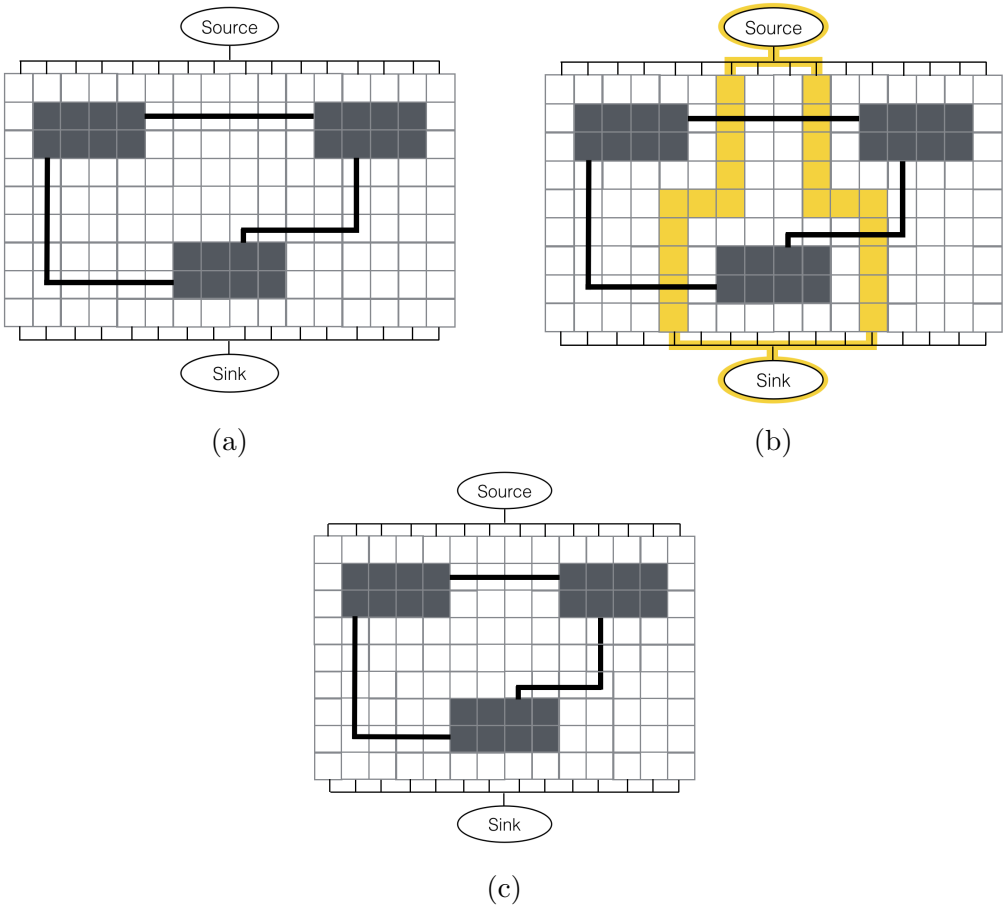


Figure 4.3: (a) A placed and routed mVLSI netlist (b) two nonlinear seams identified for removal ($\Delta = 1$); (c) the smaller chip after seam removal.

4.5.2 Perpendicular Channel Segments

Non-linear seam carving requires special handling of channel segments that run perpendicular to the carving direction. Without loss of generality, assume that we are carving in the y-direction and consider a horizontal channel segment r_i having $y_{i,t} = y_{i,l}$. A seam can be identified that cuts through r_i in such a way that its removal causes $y_{i,t} \neq y_{i,l}$; the updated channel would require a diagonal connection, or a small bend (necessitating three new channel segments), neither of which is problematic, per se.

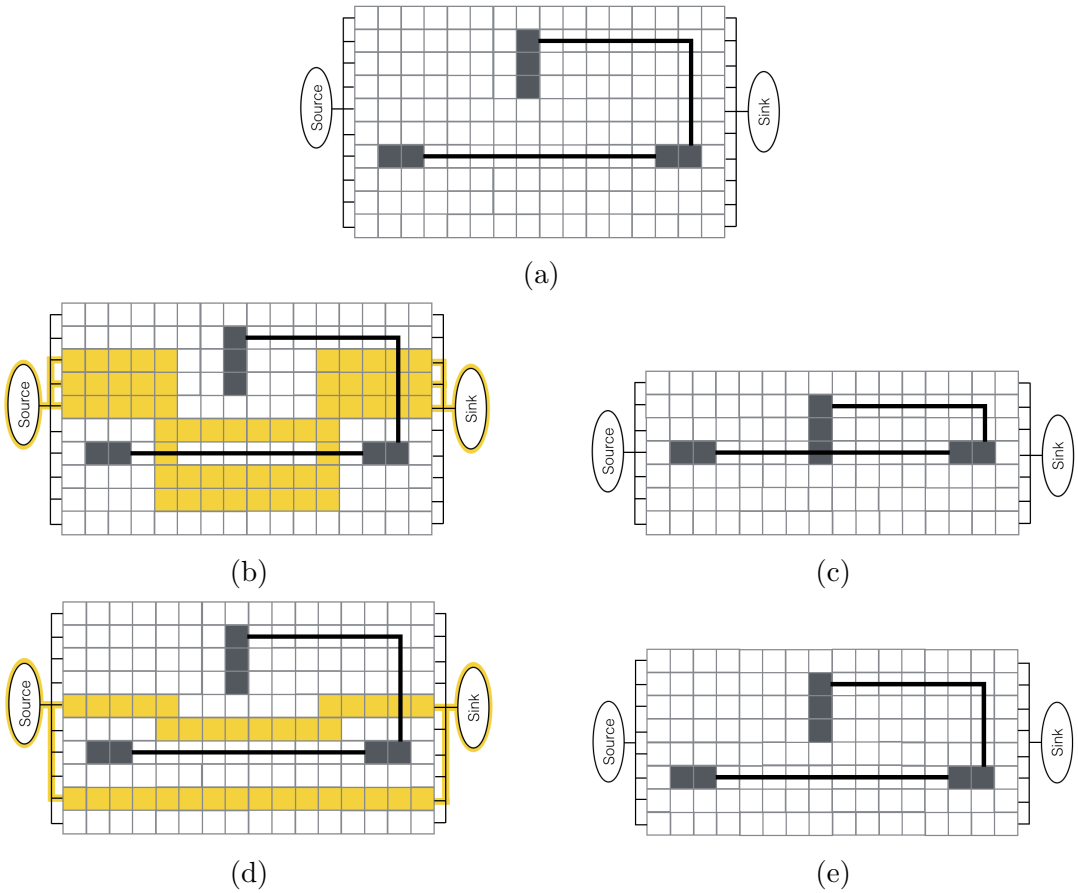


Figure 4.4: (a) A placed and routed mVLSI chip; (b) when carving along the y-axis ($\Delta = 1$), a set of non-linear seams are found that cross a perpendicular (horizontal) segment; (c) removal of the preceding seams yields an invalid layout; (d) non-linear seams are prevented from crossing the perpendicular segment; (e) removal of non-linear seams that do not cross the perpendicular segment yields a legal layout.

Perpendicular carving, however, can cause a component to shift and collide with the perpendicular channel. For example, Figure 4.4a shows a laid out mVLSI chip. Figure 4.4b shows multiple seams that cross perpendicular channel segments; in Figure 4.4c, carving these seams causes a component to shift and collide with the perpendicular channel. To prevent this, non-linear carving may not carve through perpendicular channel segments

by setting $G[x_{i,t}][z] = False$ for $y_{i,t} < z < y_{i,l}$; Figure 4.4d depicts a valid set of seams, and Figure 4.4e shows the resulting collision-free mVLSI chip after carving.

4.5.3 Seam Carving

Non-linear seam carving must choose whether to move a component or channel segment endpoint based on the opposite axis along the seam. When carving along the x-axis, any component $c_i \in C$ that exists to the right of a seam with $x_i > a_j$ between $b_j \leq y_i \leq d_j$ for any segment $s_j \in S$ will be shifted left to fill the the space that has been carved; to do this, set $x_i = x_i - 1$. All channel segments $r_i \in R$ with a source to the right of the seam with $x_{i,t} > a_j$ and $b_j \leq y_{i,t} \leq d_j$ will be shifted left to $x_{i,t} = x_{i,t} - 1$; all segments with a sink to the right of the seam with $x_{i,l} > a_j$ and $b_j \leq y_{i,l} \leq d_j$ will be shifted left to $x_{i,l} = x_{i,l} - 1$. This process then repeats similarly along the y-axis.

4.6 Experimental Results

Our Baseline algorithm is the mVLSI flow-layer Planar Placement and Network-flow based router described in Chapter 2. We implemented the Baseline algorithm in C++, along with linear and non-linear seam carving as post-processing steps. For evaluation, we use a suite of nine benchmarks: aquaflex-3b and aquaflex-5a (proprietary netlists provided by Microfluidic Innovations LLC), a bead-based HIV1 immunoassay (hiv) [34], a molecular gradients generator (mgg) [52], and five synthetic netlists. Additional information on all these benchmarks can be found in Section 6.3.2 and Table 6.2. Our implementation uses a

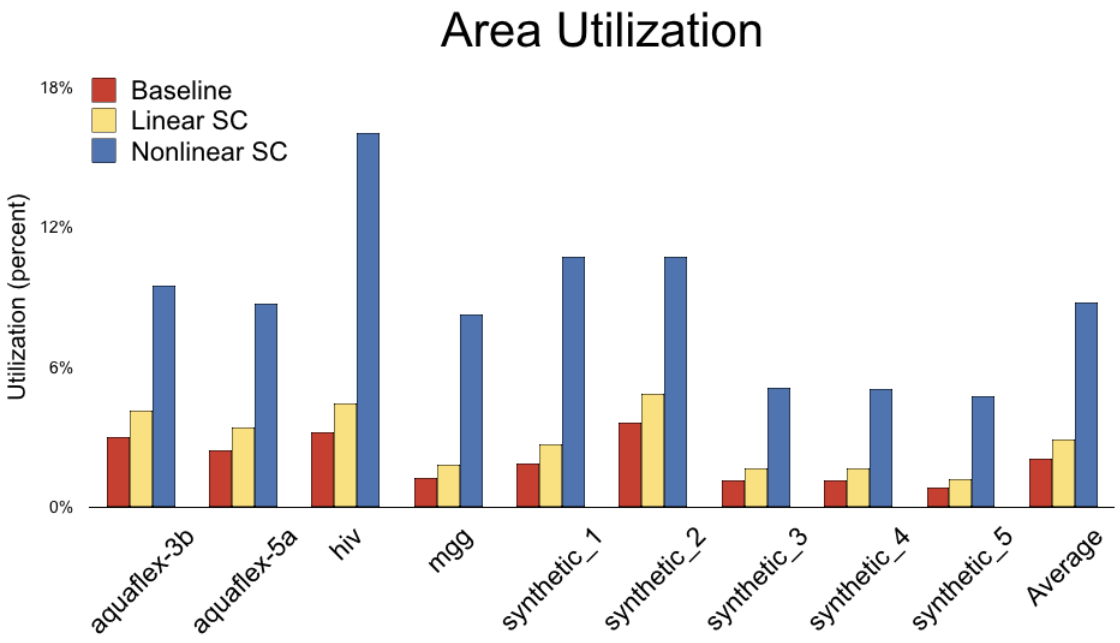


Figure 4.5: Area utilization (larger is better)

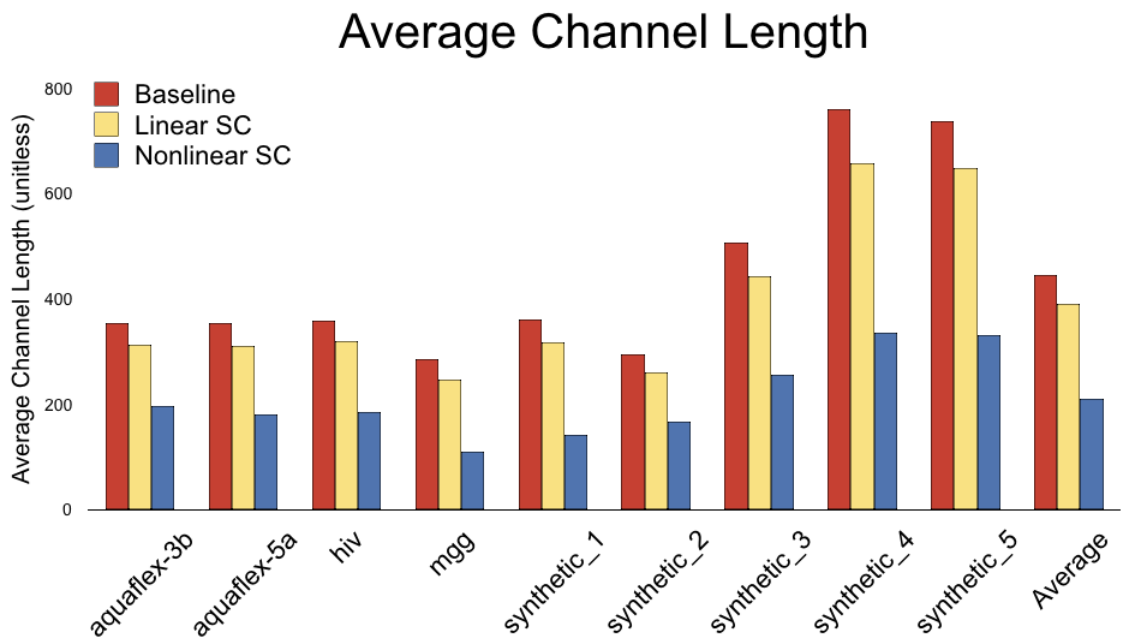


Figure 4.6: Average channel length (smaller is better)

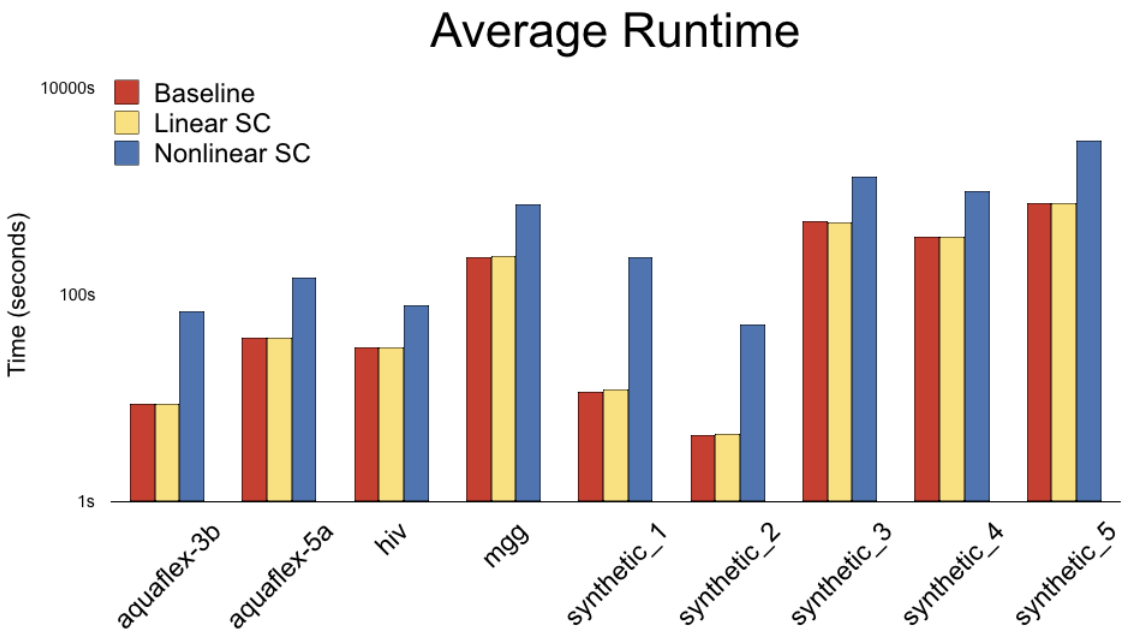


Figure 4.7: Algorithmic runtime in seconds (log. scale)

"unitless grid" with a standard buffer size $\Delta = 5$ for all benchmarks. We report the area utilization, (the percentage of the chip area consumed by components; Figure 4.5), average channel routing length (Figure 4.6), and the average algorithmic runtime across five runs per algorithm/benchmark (Figure 4.7).

Compared to the Baseline placement, linear seam carving marginally improved area utilization and average channel length, while non-linear seam carving yielded far more significant improvements. The Baseline placer is ineffective because its underlying planar graph embedding algorithm does not try to minimize area, and further loses efficiency as vertices (points) are expanded into 2D components, which necessitates further shifting of components and re-routing of flow channels to eliminate overlap. Seam carving can

effectively counteract these inefficiencies, and the results clearly show that there are far more non-linear seams available for removal than linear seams.

The runtimes reported in Figure 4.7 include the Baseline placer in all cases. Although its effectiveness is limited, linear seam carving imposes negligible runtime overhead; in contrast, the runtime of non-linear seam carving is inversely proportional to the density of the design, and, as a post-processing step, it often runs longer than the Baseline placer (e.g., synthetic_1 and 2).

4.7 Conclusion

As a post-processing algorithm, seam carving can be applied to device designs that have been physically laid out, either algorithmically or by hand. Seam carving can significantly reduce device area and channel length, making it particularly useful for the conversion of laboratory prototypes to more efficient designs that will be mass manufactured. Seam carving is quite versatile, as it is always possible to introduce and apply new carving rules to meet evolving needs. For example, if a channel is initially designed to be a certain length in order to perform its function (e.g., passive mixing) then that connection could be made invalid for carving; on the other hand, if the channel is initially longer than necessary, it could be carved repeatedly until the reduction meets the required length. We hope to further expand this work to remove assumption of rectilinear routing, as routes occurring at any angle are quite common and can often yield desirable fluid flow properties at the microfluidic scale.

Chapter 5

Automated Arraying of Subsystems via Seam Insertion

5.1 Introduction

Here we introduce a second post-processing method to increase the parallelism and/or throughput of a microfluidic device. This method can be applied either to a placed and routed device (generated through an algorithm or by hand) or to a graphical design system. The designer specifies a sub-region of the device that they would like to array (i.e., replicate k times in parallel), then the algorithm automatically replicates the subsystem, expands the requisite fluid channel subsystems to deliver fluid to the replicated components, and if needed, extends the control interface to enable lock-step single instruction multiple data (SIMD)-parallel execution of replicated subsystems.

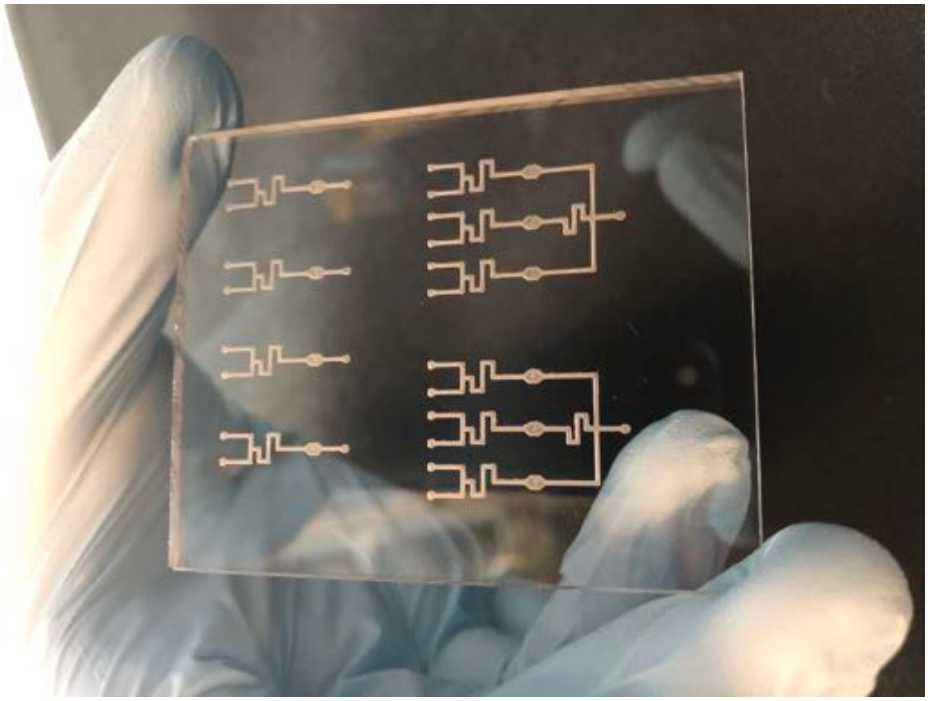


Figure 5.1: Four fabricated copies of a two input fluidic mixer device (left) and two fabricated copies of the same device after automated arraying is performed.

In a graphical design framework, this algorithm provides freedom to the designer, as opposed to placing the optimization under algorithmic control, since it is straightforward to revert back to the original design. For example, the designer can choose different subsystems to replicate and vary the replication factor k; this provides the designer with incremental control over the process. As designers gain comfort with semi-automated *design acceleration* techniques of this type, they will increasingly become amenable to increasing amounts of automation. In summary, research on microfluidic design automation will have the greatest possible impact, especially short-term, by supplementing rather than trying to replace the design tools and methodologies that are presently in use today.

Table 5.1: List of variables referenced during the seam insertion and automated arraying method description

Variable	Description
A	Input microfluidic architecture
C	Set of placed components
R	Set of routed connections
m	The width of the input microfluidic architecture
n	The height of the input microfluidic architecture
P_i	The (x,y) coordinate of the ith component
w_i	The width of the ith component
h_i	The height of the ith component
$P_{i,s}$	The (x,y) source coordinate of the ith connection
$P_{i,t}$	The (x,y) sink coordinate of the ith connection
Δ	Optional component buffer space
δ	Amount of space to be inserted
G	The grid of boolean nodes for seam identification
S	A set of straight line segments representing a seam
d	The seam identification direction (either north and south or east and west)
$P_{i,a}$	The (x,y) start coordinate of a seam segment
$P_{i,b}$	The (x,y) end coordinate of a seam segment
B	The bounding box representing the users arraying selection
N, E, S, W	The north, east, south, and west edges of the bounding box
$P_{(n,e,s,w),s}$	The (x,y) source coordinate of the edge N, E, S, W
$P_{(n,e,s,w),t}$	The (x,y) sink coordinate of the edge N, E, S, W
$X_{(n,e,s,w)}$	The connection segments $r_i \in R$ that cross the edge N, E, S, W
S_c	The set of components within B
S_r	The set of connection segments within B
α	The width of the selection B
β	The height of the selection B
a	The array direction (either north and south or east and west)
k	The number of desired selection replications
γ	The desired buffer space between the arrayed selections

5.2 Seam Insertion

The input to seam insertion is a placed and routed microfluidic very-large-scale integration (mVLSI) architecture $A = \{C, R, n, m\}$, where C is a set of placed components, R is a set of routed connections, and n and m are the respective height and width of the layout. We represent each microfluidic component $c_i = (P_i, w_i, h_i)$ using a bounding box: point $P_i = (x_i, y_i)$ is the upper-left corner of the component boundary box and h_i and w_i are its respective height and width. Each routed connection segment $r_i = (P_{i,s}, P_{i,t})$ is a straight-line connection between points $P_{i,s} = (x_{i,s}, y_{i,s})$ and $P_{i,t} = (x_{i,t}, y_{i,t})$; multiple segments may comprise a longer channel with twists and bends. The physical layout process may include an additional parameter, Δ, which adds buffer space around each component to improve routability or meet fabrication requirements. Finally, a user input value δ is required which represents the amount of space that should be inserted into the system.

In microfluidic devices all space in an architecture can be classified into three distinct categories:

1. Component space which cannot be removed without disrupting the functionality of the component.

2. Connection space which represents a flow or control channel connecting two components, and can be of any length as long as there is a continuous path between both components.

3. Empty space which is unused except as buffer space for components in the case where $\Delta > 0$.

A seam is defined as a path through the architecture that travels from one perimeter edge to the opposite perimeter edge and contains no points which would affect the functionality of the microfluidic device; this ensures that correct device functionality is maintained when the seam inserts additional space in the design. Invalid points include any space existing within a component as well as any connection segments which run parallel to the seam identification direction d.

5.2.1 Grid Creation

The setup to this process is the same as the method introduced in Chapter 4. We begin by creating an $m \times n$ boolean grid G that represents the placed and routed microfluidic device. All the grid values are initially set to to $True$.

For each component $c_i \in C$ at position $P_i = (x_i, y_i)$ we set $G[j][k] = False$, $x_i \leq j \leq x_i + w_i, y_i \leq k \leq y_i + h_i$. This disallows any points within a component from being selected as part of a seam. Additionally, a subset of the connections segments must also be disallowed for seam selection.

If a seam is identified that crosses through a connection segment that runs parallel to the seam identification direction, then when the seam shifts the architecture the source and sink of the segment would no longer be parallel. This shift in connection segment angle can cause it to collide with other components in the system, making the architecture invalid. Because of this, connection segments that run parallel to the seam identification direction are disallowed for seam selection. That is, when d is north and south, for all segments $r_i \in R$ where $P_{i,s}.x = P_{i,t}.x$, $G[P_{i,s}.x][z] = False$ for $P_{i,s}.y < z < P_{i,t}, y$. When d is east and west, for all segments $r_i \in R$ where $P_{i,s}.y = P_{i,t}.y$, $G[z][P_{i,s}.y] = False$ for $P_{i,s}.x < z < P_{i,t}.x$.

5.2.2 Seam Identification

Once the grid has been initiated, we add two additional node *source* and *sink* to either the north and south edges of the grid ($g \mid g \in G, g.y = 0$ and $g \mid g \in G, g.y = n - 1$) or the east and west edges of the grid ($g \mid g \in G, g.x = 0$ and $g \mid g \in G, g.x = m - 1$) depending on d.

A seam S, which is a collection of straight line segments $s_i \in S$ such that $s_i = (P_{i,a}, P_{i,b})$ and $P_{i,a}$ and $P_{i,b}$ are the (x, y)-coordinates of the two endpoints, is then identified. This search must begin at the *source* node and end at the *sink* node, but can traverse through any other $True$ value nodes in G. The path is identified through the grid using a shortest path algorithm, such as Lee's algorithm [33], from *source* to *sink*. If the seam needs to traverse through a specific node or set of nodes in the grid, then the single shortest path can be broken into a number of shortest path searches. A shortest path is then found between each closest pair of points (sorted by either x for east and west or y for north and south seam identification direction), and the collection of shortest paths are then combined to create a single continuous seam.

5.2.3 Buffer Insertion

After a seam has been identified, it is used to insert δ additional space into the architecture. Assuming a north and south seam identification direction d, any component $c_i \in C$ that exists to the east of a seam with $P_i.x > P_{j,a}.x$ between $P_{j,a}.y \leq P_i.y \leq P_{j,b}.y$ for any segment $s_j \in S$ will be shifted east by δ to add additional space into the design. To do this, set $P_i.x = P_i.x + \delta$. All channel segments $r_i \in R$ with a source to the east of the seam with $P_{i,t}.x > P_{j,a}.x$ and $P_{j,a}.y \leq P_{i,t}.y \leq P_{j,b}.y$ will be shifted east to $P_{i,t}.x = P_{i,t}.x + \delta$; all

segments with a sink to the east of the seam with $P_{i,l}.x > P_{j,a}.x$ and $P_{j,a}.y \leq P_{i,l}.y \leq P_{j,b}.y$ will be shifted east to $P_{i,l}.x = P_{i,l}.x + \delta$. If the seam identification direction is east and west, then the x and y coordinates are swapped, and everything is shifted south by δ.

5.3 Automated Arraying

5.3.1 Input

As input, this method requires a fully placed and routed mVLSI architecture $A = \{C, R, m, n\}$ as previously described in section Section 5.2. Additionally, a user selected portion of the device represented by the bounding box $B = \{N, E, S, W, S_c, S_r, \alpha, \beta, a, k, \gamma\}$ is required as input. N, E, S, W are the north, east, south, and west edges of the bounding box respectively. The sets S_c and S_r represent the components and connection segments that are contained entirely within the selection, and α and β represent the width and height of the selection, respectively. a represents the array direction input from the user, which must be either north and south or east and west. k represents the number of times the selection should be replicated and γ is an optional parameter representing additional buffer space that should be inserted between replications during arraying. $N, E, S, W = (P_{(n,e,s,w),s}, P_{(n,e,s,w),t}, X)$ such that $P_{(n,e,s,w),s}$ and $P_{(n,e,s,w),t}$ are the start and end points of a bounding box edge and X is the set of connection segments that cross that particular edge. For all connections $r_i \in R$, if a segment $(P_{j,s}, P_{j,t}) \in r_i$ crosses an edge $N, E, S, W \in B$ then it should be added to the set X for the edge that it crosses.

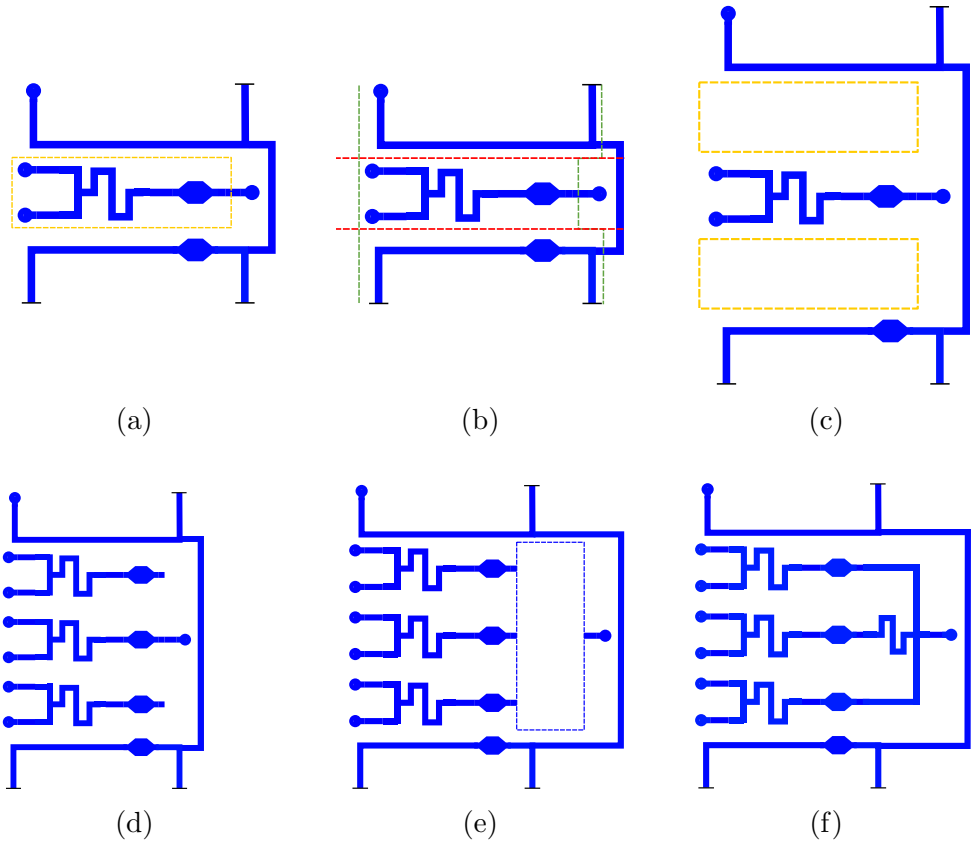


Figure 5.2: (a) A placed and routed device is used as input, with a portion of the device selection for arraying (yellow). The input device may contain other components and/or connections that are not part of the selection but must be accounted for when performing the replication to avoid overlapping or intersections. The selection is validated to ensure that no components cross the selection. (b) Two sets of seams are identified, one running perpendicular to the array direction (green) and another running parallel (red). (c) The parallel seams are used to insert additional space (yellow) equal to the size of the selection (height in this example) based on the number of replications k that need to be inserted. (d). Components and connection segments within the selection are replicated and shifted into the newly created space. (e) The perpendicular seams are then used to insert additional space (blue) equal to the size of of a junction that can accommodate the number of arrayed selections, and (f) an appropriate junction is inserted into the newly created space.

5.3.2 Selection

Some considerations must be made when choosing a selection. In the selection, all components $c_i \in S_c$ must resides within the bounding box B and no component may cross any

edge $N, E, S, W \in B$. This would represent only a piece of a component being selected for arraying which would not be able to function properly. If the system allows composite components, which are collections of components and connections represented as a single higher order component, then that composite component should first be reduced into its individual pieces before checking this criteria.

Additionally, any connection segments that cross the selection in parallel to the array direction a must traverse the entire selection through to the opposite edge of the one it enters. If a connection segment enters one edge of the selection but does not continue to the other, then the replicated versions will not be able to connect to the rest of the device and will not function properly. If the connection segment is incidental to the selection and not necessary for the selection to function properly, then it can be annotated by the user as an *excluded connection* and will not be processed for replication.

5.3.3 Breaking Crossing Connections

Connection segments that cross the selection must necessarily exist partially within the selection, which should be arrayed, and partially outside the selection which should not. Each crossing connection segment $r \in X_{(n,e,s,w)}$ is split into two separate connection segments r_i (inner segment) and r_o (outer segment). The connection segment $(P_{i,s}, P_{i,t})$ from $r \in X_{(n,e,s,w)}$ that crosses the selection edge is identified and is split at the point P_x where the edge N, E, S, W and $(P_{i,s}, P_{i,t})$ intersect such that $r_i = (P_{i,s}, P_x)$ and $r_o = (P_x, P_{i,t})$, assuming $P_{i,s}$ exists within the selection and $P_{i,t}$ exists outside. If both points exist outside of the selection, then the connection must cross two opposite edges and the "internal"

point is considered to be the intersection point with the other edge. After splitting all the segments, the inner portions (r_i) are added to S_r.

5.3.4 Control Layer Considerations

There are two cases to consider when control connections cross the selection edges. The first is connections that pass through the selection perpendicular to the arraying direction a and crossing two opposite boundary edges, but are incidental to the selection. This can occur in designs with dense routings where flow or control connections that are only needed to run components outside the selection cannot be spatially excluded from the it, and therefore do not need to be replicated because the components they control are not being replicated. These control connections should be annotated as excluded connections by the user and will not be added to S_r or any $X_{(n,e,s,w)}$.

The second consideration is control connection segments that run parallel to the array direction and need to connect between the arrayed units to allow for SIMD style parallel execution of the arrayed components. If the control connections enter and exit the selection at parallel points then they can simply be arrayed in the same manner as the flow connections. This is because when the entry and exit points of the connection are aligned, the output from one selection becomes the input to the next selection after replication, and all valves connected to that control line will be actuated in parallel. If selection buffer space γ is introduced between the selections then the control lines can be directly extended. If the input and output positions are not parallel then a reclamation step must be utilized to bridge the difference between the two positions. This requires enough selection buffer space

γ to be inserted to perform a local routing from one selections outputs, through the buffer space, to the inputs of the next.

5.3.5 Replication and Arraying

Once the crossing connection segments have been broken and the selection finalized, additional space must be inserted to allow the selected connection segments and components to be replicated and integrated into the system without introducing intersections or design rule violations.

This process begins by finding two seams through the system. For a north and south array direction a, seam identification is performed in the opposite direction (d = east and west) and using the north and south selection edges as intermediate routing points. A seam segment is found from the *source* node to $max(P_{n,s}, P_{n,t})$ and from *sink* node to $min(P_{n,s}, P_{n,t})$. A final segment $(P_{n,s}, P_{n,t})$ representing the edge itself is then added to create a full seam. This process is then repeated for the south edge to create a set of two seams. For an east and west array direction, seam identification is performed in the north and south direction d and utilize the east and west selection edges to find seam segments from *source* to $max(P_{e,s}, P_{e,t})$ and from *sink* node to $min(P_{e,s}, P_{e,t})$ adding $(P_{e,s}, P_{e,t})$ as the final segment. This is then repeated using the west edge to create two seams.

These seams then insert δ additional empty space that will be used for replicating the components and connection segments in the selection B. The value δ is determined by the number of selection replications, k, that need to be added to the system, the width α or height β of the selection, and any selection buffer γ. If there are an odd number of replications (including the original, such that $k = 1$ would yield the original design) then

$((k-1)/2) * \alpha + \gamma$ or $((k-1)/2) * \beta + \gamma$ space would be inserted into both seams in the east and west or north and south direction respectively. If there are an even number of replications then $(((k/2) * \alpha) - (\alpha/2) + \gamma$ or $(((k/2) * \beta) - (\beta/2) + \gamma$ is added north or west and south or east directions. Additionally, for an even number of replications the original selection is shifted by $(\alpha/2) + \gamma$ in the east and west case or $(\beta/2) + \gamma$ in the north and south case.

Once the space has been allocated the components and connection segments within the selection must be replicated and arrayed over the new space. A copy of each component $c \in S_c$ and each connection segment $r \in S_r$ within the selection B is created and initially set to the same position as the original selection. Each replication is then shifted $(i-1)*(\alpha+\gamma)$ or $(i-1) * (\beta + \gamma)$ either north and east for $i - 1 < (k-1)/2$ or south and west for $i - 1 \geq (k-1)/2$ based on its index i (where $i = 1$ is the original selection) and array direction.

5.3.6 Junction Insertion

Once the selection has been replicated, junctions must be inserted to connect all the replicated inner connection segments that crossed a bounding edge to the single outer segment of the selection that was not arrayed. To facilitate this seams are found in the same manner as Section 5.3.5 but using the inverse d value and the inverse set of bounding edges.

Space is then inserted using a δ value equal to the width in the case of an east and west array direction a, or height in the case of a north and south arraying direction a of a large enough junction to support the the number of replications in the device technology

and at least as large as the total selection space after replication to allow it to reach all selection inputs. The required junction can then be inserted into the newly created space.

5.3.7 Junction Selection

There are several factors that must be considered when determining what junction should be inserted to connect the system external to the arrayed selection to each selection. Junctions must be introduced on each selection edge parallel to the array direction that has one or more crossing connection segments. Each junction must contain a number of "inputs" on one side equal to the number of crossing connection segments of the selection edge it will abut ($|X_{(n,e,s,w)}|$), and a number of "outputs" on the opposite side equal to the number of crossing connection segments multiplied by the replication number ($k * |X_{(n,e,s,w)}|$).

Additionally, a consideration needs to be made as to whether the junctions being introduced should be *passive junctions*, containing no valves and driven by channel length and geometry, or *active junctions*, containing valves and being driven by pneumatic pressure from the control layer.

Passive junctions contain no valves, and therefore require no additional processing of the control layer or the microfluidic application. This, however, means that they will allow for the fluids in the flow channels to mix freely as observed in Figure 5.3b and Figure 5.3d. Active junctions contain valves and therefore must contain control lines to drive them. These valves allow for fluids to be actively routed from an arbitrary input line to an arbitrary replicated selection input, but will require at minimum a modification to the application as the actuation of the junction will need to be accounted for.

Active junctions can be further classified into two different types, *embedded control* junctions and *routable control* junctions. Embedded control junctions, as illustrated in Figure 5.5b, embed the pneumatic inputs necessary to drive the junctions control layer within the component itself. This has the benefit of allowing for the active junction to be added to the system without the need to re-route the control layer. Routable control junctions, instead of containing pneumatic inputs, route the control lines to the edges of the junction as illustrated in Figure 5.5c. This means that an additional routing step will be required to connect the new junction to an existing control port. This method is best suited for devices that use templated input and output port positions, such as cartridges that need to be compatible with a standard control device, or where a control optimization step is desired and the entire control layer will be re-optimized and re-routed.

In all these cases, it is assumed that the designers has access to a library of available junctions for their fabrication technology or are able to generate the appropriate junction as necessary using a previously defined methods [55, 18]. Because junctions vary depending on type and fabrication method, junction generation is considered out of scope for this work.

5.4 Case Studies

5.4.1 Proof of Concept

As a proof of concept for this new technique, a new passive microfluidic device was designed by hand, fabricated, and tested. The same design was then arrayed, fabricated, and tested to show that the functionality from the original design was preserved and could be performed

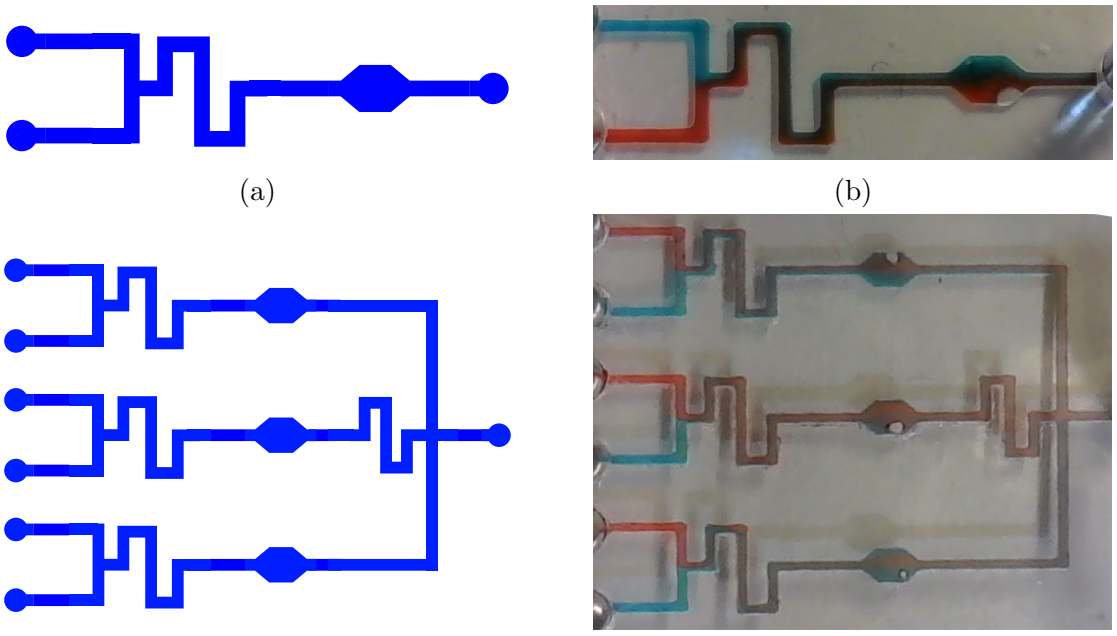


Figure 5.3: (a) The design for a two input mixer with a detection cell before the output and (b) the fabricated version of that device showing red and blue food coloring being passively mixed to create purple. (c) The same device after being arrayed north to south with a $k = 3$ replication number and (d) the same experiment being re-run and showing the same mixing.

on all of arrayed selections. The newly designed device contains two fluid inputs, each connected to a single serpentine channel designed for passive mixing, which can be seen in Figure 5.3a. Once the fluids are mixed they flow to a larger detection chamber to aid in external sensing before being removed through an output port.

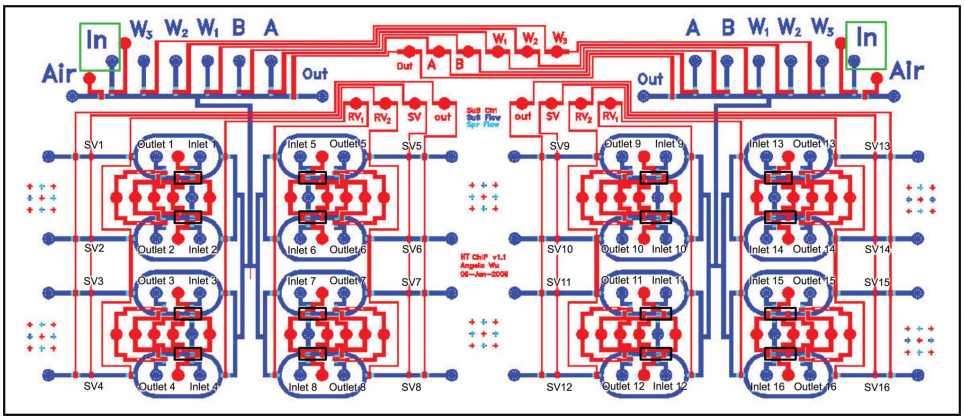
This device was created as an scalable vector graphic (SVG) using Inkscape and fabricated in cast acrylic using a computer numeric controlled (CNC) mill with 0.5μm flow channels. Two fluids (red and blue food coloring) were input and mixing was observed in the serpentine channel as seen in Figure 5.3b, creating a purple output fluid. This mixed fluid then traveled to the detector chamber before exiting the system through the output

port. Once the original device was validated, the arraying algorithm was applied to a selection that included all components except the fluid output, as illustrated in Figure 5.2a. This created a new design (Figure 5.3c) that contains three independent two input mix and detect stages that collect into a single passive junction and collectively flow to a single fluid output.

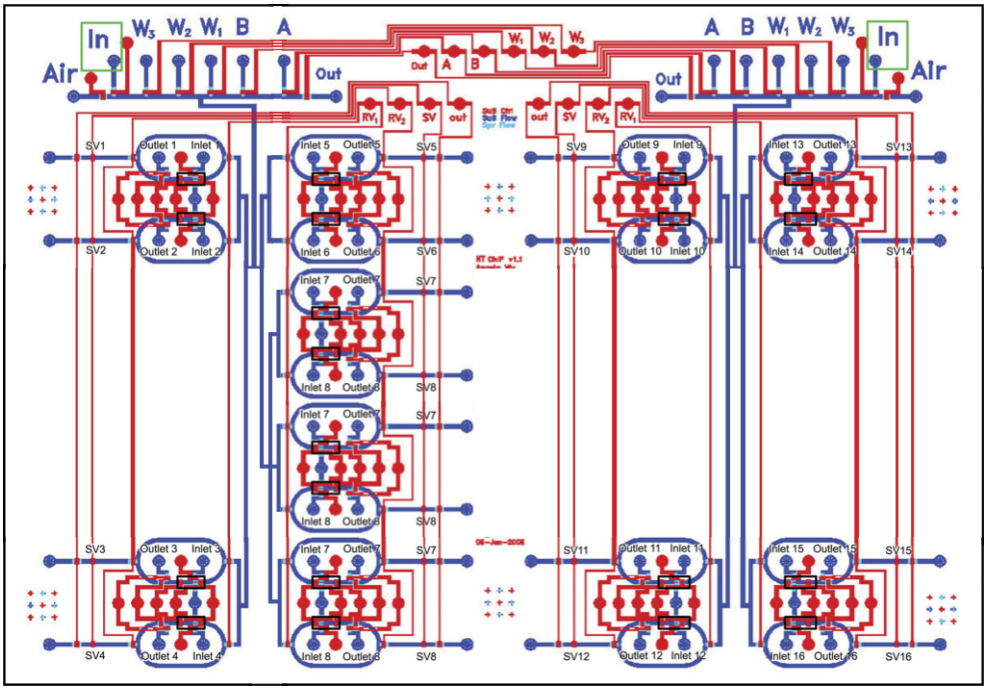
The arrayed device was then fabricated in the same manner as the original device and tested with three pairs of the same fluids (red and blue food coloring). All three of the mix and detect stages were capable of generating mixing in the serpentine channel similar to the original device, as seen in the purple output fluids in all three detectors in Figure 5.3d.

5.4.2 HT-CHiP Arraying

The high throughput automated chromatin immunoprecipitation (HT-ChiP) device originally designed by Wu et al. is capable of screening 16 different protein-DNA reactions simultaneously [67]. This device and other chromatin immunopreceipitation devices are typical of benchmarks used when testing many microfluidic placement and routing algorithms [59, 71], especially when illustrating how placement and routing algorithms are able to scale to larger sizes because of the devices relatively heterogeneous design. This device, illustrated in Figure 5.4a, contains sixteen circular mixer units each of which contains an independent fluid input, with each pair of mixers using a a shared set of pneumatic control ports. Additionally, banks of eight mixers are configured to share a single connection to a set of seven fluidic inputs and an output.



(a)



(b)

Figure 5.4: (a) The high throughput chromatin immunoprecipitation (HT-CHiP) designed by Wu et al. [67]. (b) The automated arraying technique is applied to one mixer bank (which consists of two mixers units with shared control) from the original design with a replication value $k = 3$ to insert two additional mixer banks.

From the original design a pair of mixer units, their independent outputs, and their shared pneumatic control ports were selected for arraying with a replication value of

$k = 3$. Since the mixers existing control lines were aligned and actuated in a SIMD fashion they can be arrayed directly and can be actuated using the original application. The only modification to the application that needs to be accounted for after the device is arrayed is to actuate the two new sets of pneumatic control inputs that were arrayed automatically by virtue of being contained within the selection. If purely passive devices had been contained in the selection, then no modification to the application would be necessary.

It should be noted that the selection made in this case was done for illustrative purposes. Normally, an entire cross section of banks would be selected for arraying which would fill in the relatively empty portions of the design with additional mixer units, bringing the total mixer unit count to thirty-two rather than the twenty seen in the example.

5.4.3 Fluidic Memory Insertion

Any flow connection, which is an etched channel in a fabricated device, can be used to temporarily store fluids for later use. This concept is best illustrated in the general purpose microfluidic device introduced by Urbanski et al. [61] where fluidic memory was embedded into the system to allow for complex mixtures to be made from a large number of base chemical and biological fluids without the need for each to have its own independent fluidic input.

Using automated arraying any straight line connection segment long enough to hold the desired volume of fluid can be automatically converted into fluidic storage. First, a selection of the channel (Figure 5.5a) is made and the replication number k is set to the number of fluids that need to be stored in parallel. This number may need to be increased

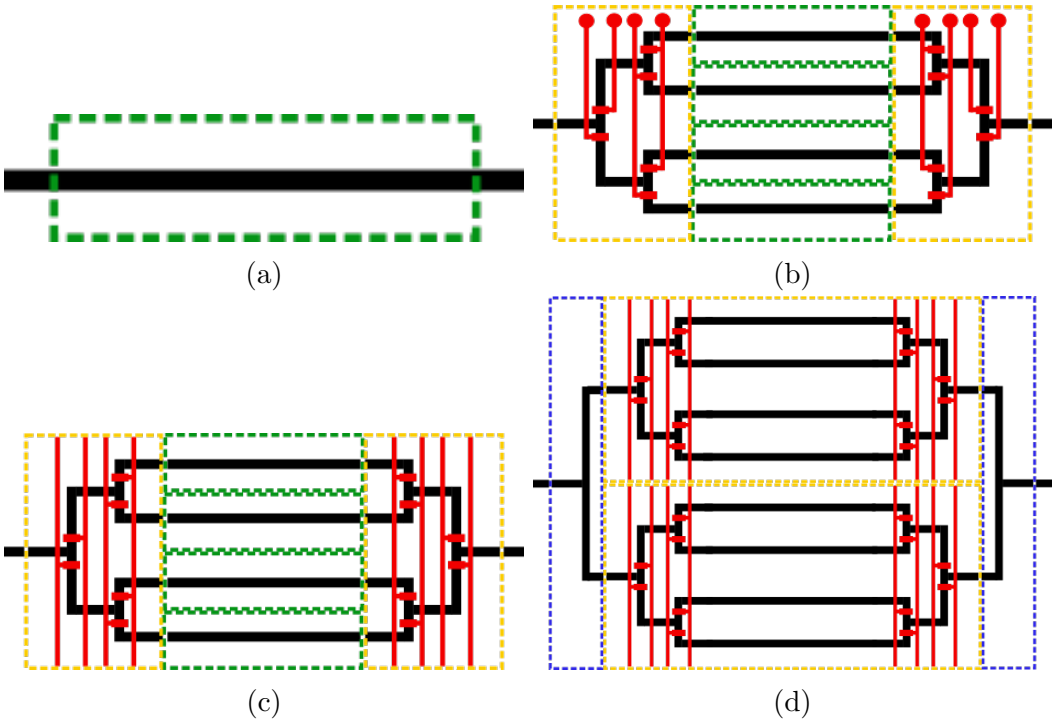


Figure 5.5: (a) An arbitrary channel segment in a placed and routed device (green) is selected and (b) arrayed using the automated arraying technique presented here. Junctions are added (yellow) to allow the new channels to be used as storage. The introduced junctions can either (b) be embedded control or (b) routable control junctions. (d) These fluidic memories (yellow) can then be re-arrayed as necessary, creating more memory and adding additional junctions (blue).

by one if a bypass line is needed to allow fluids to pass through without affecting the other fluids being stored. From there the arraying step will create $k-1$ number of new channel that can be used to store fluids. Two junctions are then automatically inserted, however an active junction must be used. Because the fluidic memory will be required to select fluids from individual channels on demand for use in the system, each channel needs to be individually addressable which can only be accomplished in microfluidics with active valving.

The active junction that is inserted can either be an embedded control junction (Figure 5.5b) or a routable control junction (Figure 5.5c). Which version of the junction should be used depends largely on the original device. If the original device was a totally passive device, then the embedded control junction would likely be chosen because no additional processing steps would need to be performed except for the creation of a new microfluidic application, since control is now required to run the device. If the original device was active, then it may be preferred to use a routable control junction so the newly introduced control lines can be routed to previously defined inputs, reducing the number of total inputs to the device.

Once fluidic memory has been introduced into a device if more memory channels are needed then the original device can be re-arrayed with a higher replication value k. Alternatively, the memory that has already been introduced can be arrayed, as illustrated in Figure 5.5d. Here, the memory from Figure 5.5c has been selected and the automated arraying method is performed on it with a replication number $k = 2$.

5.5 Conclusion

The algorithm presented in this chapter solves the time consuming and error prone task of automatically arraying a subsection of a microfluidic device, primarily in order to increase throughput. In order for the arrayed device subsections to function correctly, the fluidic resistance of all arrayed subsections must be equalized, as this chapter illustrated in the small serpentine addition to the middle path in Figure 5.3c. This detail is often overlooked in manual design leading to errors that are only discovered during testing. Utilizing automated

arraying in conjunction with a multiplexer generator ensures that this class of errors does not occur. While the multiplexer generation is only briefly mentioned here, the generation of functionally correct junctions and multiplexers for a variety of different microfluidic technologies and applications is an area ripe for further study. In the future, we hope to partner with domain experts to develop a robust multiplexer generator toolkit which can be used here and in any other design toolchains or workflows.

Chapter 6

ParchMint: A Microfluidics Benchmark Suite

6.1 Introduction

The project that this dissertation is based around was originally started by Dr. Jeffrey McDaniel in 2012 to support the creation of the simulated annealing-based method for component placement [41] which has been referenced throughout this document. While the project was developed to outlive that specific publication, the benchmarks that were originally used for publication were based on a set of openly available digital microfluidics benchmarks created by Technical University of Denmark (DTU) [22] as that was one of the few available sets of benchmarks and had already been used to generate continuous-flow architectures by Minhass et. al [45]. When planar embedding based methods were first being investigated during the development of the Planar Placement and Network-

flow based routing method presented in Chapter 2 as a new placement method to that same project, some of the benchmarks that were already available were found to be non-planar. Since planarity is a requirement to that method the original benchmarks were hand planarized through the introduction of additional switch components. This original set of hand planarized benchmarks, along with a small subset of additional benchmarks created from literature and those provided by Microfludic Innovations LLC, were used for testing all the methods presented in the previous chapters.

As the project evolved and there was a desire to move towards fabrication as a form of validation for algorithmically placed and routed devices, as well as devices that were modified using post-processing methods, the benchmark set became a roadblock. While the benchmarks that had been constructed were excellent for testing how the algorithms scaled along with devices, they were too large for us to reasonably fabricate and test with the resources we had available. To solve this problem in the short term, we started to develop small architectures with accompanying entity designs to allow us to develop a full design capable of fabrication, with the first result of this effort being the mixer device in Chapter 5 which was subjected to the automated multiplexing post-processing method. Entities were created specifically for that design, and while the netlist was laid out by hand for the sake of image quality, the automated multiplexing method was applied directly to that netlist to create a file which was then fabricated and tested.

While this method is effective at creating high quality components which we know will fabricate well in our facilities, there are two main issues. The first is that this is a time and labor intensive process that requires many cycles of development, fabrication, and

testing before valid components and devices can be created. The second is that developing components which are well fabricated in our facilities and of interest to our team and collaborators means we are only creating benchmarks for a specific subset of microfluidics technologies, which may skew our development efforts to target those specifically without being generalizable to the field. In order to attempt to scale our benchmark development more quickly and try and correct for choice bias, we reached out to Cross-disciplinary Integration of Design Automation Research (CIDAR) labs at Boston University (BU). CIDAR labs had the inverse problem that we had in that they had originally designed their tools with fabrication in mind and had many small benchmarks along with entities they were capable of fabricating. This means that they were able to validate their designs through fabrication, but their designs were too small to know if their algorithms could scale past a relatively small number of components and connections.

Because our needs were so complimentary, we began a collaboration which evolved into a long-term project to develop a benchmark ecosystem for continuous-flow microfluidic devices. The first step in the development of this ecosystem was a JavaScript object notation (JSON) standard file format, which started with the desire to unify the benchmarks developed by ourselves at University of California Riverside (UCR) as well as the CIDAR labs team at BU through a standard interchange format in order to create a large available set of benchmarks to utilize for evaluation. This then evolved into ParchMint, which is a new system for defining, discussing, and generating continuous-flow microfluidic device benchmarks in a standard and systematic way. ParchMint contains a standard interchange format which allows research groups to exchange designs at various stages of the design

automation process, provides a standard method for verifying design specifications and the comparison of results. The associated benchmark suite is more substantial than any currently being used in literature and is highly varied across a number of known metrics of interest to design automation researchers as well as some new parameters. These parameters and their role in quantifying microfluidic designs in literature will help to deepen the discussion around the effectiveness of design automation algorithms to real-world continuous-flow microfluidic devices. Finally, we introduce Scribe, a benchmark generator that is capable of creating arbitrarily large benchmarks with a known solution that can be tuned to stress test how algorithms can cope with variance in specific metrics of interest.

The long term goal of this project is to create a benchmark ecosystem covering the evaluation of everything from layout and fabrication to functional validation and individual component design.

6.2 Background

Research into continuous-flow microfluidic microfluidic very-large-scale integration (mVLSI) design automation currently relies on a small number of architectural netlists, as illustrated in Table 6.1. These benchmarks have been generated either by converting bioassay specifications into an artificial architecture or by hand converting microfluidic designs from literature, with little consistency between research teams in the benchmark set that they use for testing. Here we briefly describe the provenance of the existing benchmarks and some of the problems associated with them.

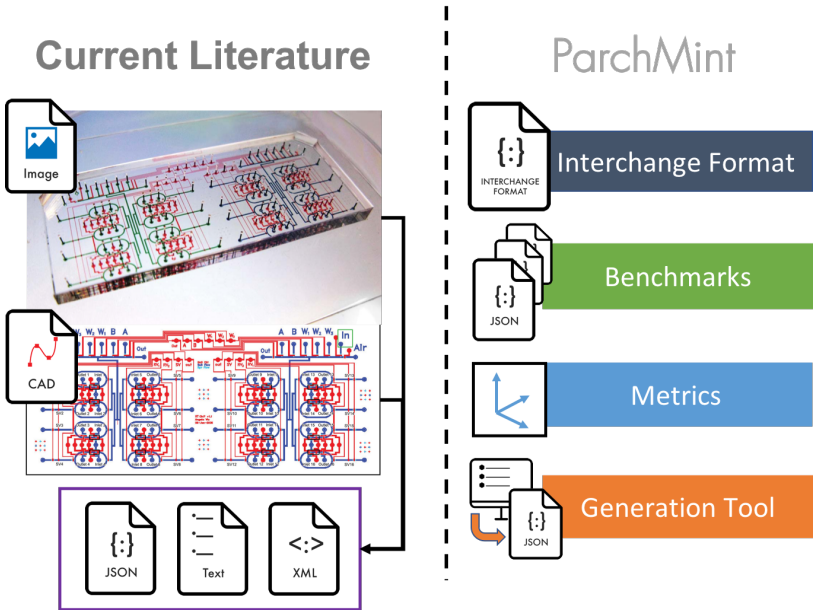


Figure 6.1: Currently, individual research teams create benchmarks by analyzing device images or layout files (such as CAD files) manually, as represented in the figure with device from Wu et. al [67], or by converting bioassay applications through a variable architectural generation step [45, 57, 48, 28]. ParchMint solves this problem by providing a standard method for specifying benchmarks and a publicly available set to test against. Additionally, it provides metrics to characterize new benchmarks and a method for automatically generating new ones.

6.2.1 Bioassay Based Benchmarks

The only benchmark sets that are currently publicly available are two suites of bioassay specifications for electrowetting-based digital microfluidics, one released by DTU [22] and another by Duke University in the United States [54]. Over the years, these have become the de facto "standard" benchmark suite for continuous-flow based microfluidics and constituting approximately half of benchmarks in literature today. Because these benchmarks represent a collection of microfluidic applications for electrowetting, not architectures for a flow-based device, they must be converted into an architectural netlist to be used for

Table 6.1: Comparison of the benchmarks used for the evaluation of different physical design algorithms in literature

Benchmark	Minhass et. al [45]	McDaniel et. al [41]	Yao et. al [71]	Wang et. al [64, 65]	Tseng et. al [59, 58]	Grimmer et. al [17]	Crites et. al [9, 10]
EA [45]*	✓						
PCR [44, 54]*	✓	✓		✓		✓	
IVD [44]*	✓						
CPA [44]*	✓						
Synthetic 1 (10) [44]*	✓	✓	✓				✓
Synthetic 2 (20) [44]*		✓	✓				✓
Synthetic 3 (30) [44]*	✓	✓	✓				✓
Synthetic 4 (40) [44]*	✓	✓	✓				✓
Synthetic 5 (50) [44]*	✓	✓	✓				✓
ProteinSplit-1 [54]*				✓		✓	
ProteinSplit-2 [54]*				✓		✓	
InVitro-1 [54]*				✓		✓	
InVitro-2 [54]*				✓		✓	
InVitro-3 [54]*				✓		✓	
Kinase act. [15]					✓		
Acid proc. [24]					✓		
mRNA iso [36]					✓		
Gradient Generator [52]							✓
HIV1 [34]							✓
ChIP (4IP) [66]					✓		
ChIP (10IP)					✓		
Chip1			✓				
Chip2			✓				
Cell free Bio Net [47]					✓		
AquaFlex-3b †							✓
AquaFlex-5a †							✓

*interpreted from a bioassay application
†Proprietary designs from Microfluidic Innovations LLC
PCR is listed in the DTU and Duke benchmark suites

continuous-flow based physical design automation. Because there are many different methods for converting an application to an architecture [45, 57, 48, 28], each research team has used a different method of generating the architecture netlists from the application. Some of these methods do not generate a planar architecture and will require further processing creating another avenue for the benchmark interpretation to diverge.

Even when there is a desire to use the same benchmarks across groups, there is no publicly available set of architectural netlists or agreed upon file format for specifying them. This makes it nearly impossible to compare previously published algorithms against newer literature in a meaningful way. For example, McDaniel et. al uses *Synthetic 1-5* when reporting results for a Simulated Annealing (SA) based placement method [41]. Wang et. al [64] builds on this work by introducing a negotiation based router and a placement adjustment system and compares against McDaniel et. al. However, when making the comparison Wang et. al uses the Duke University benchmark set rather than the DTU benchmark set used by McDaniel et. al, making a direct comparison of the two methods impossible. Compounding this issue is the lack of specific information given for each benchmark. McDaniel et. al gives no additional information on the benchmarks used, and Wang et. al only provides the number of components each benchmark contains. Without additional information, it is impossible to know if the method was truly superior, or simply better suited to that particular set of benchmarks.

6.2.2 Literature Based Benchmarks

There is a wealth of literature describing microfluidic devices that were designed manually (e.g., using AutoCAD), fabricated, and then used to perform a variety of biological assays. Some of these publications provide enough visual information, like the example in Fig. 6.1, to derive a netlist through visual inspection [61]. In more recent years, microfluidic design automation researchers have reported experimental results using this approach [48, 59]; however, to the best of our knowledge, these netlists and their corresponding file formats used in these publications have not been widely disseminated.

Even with this method of benchmark creation, there is a high amount of variability in interpretation. Because these systems represent components using a bounding box, there can be variance in the measured size of the component from the source material. Even more, fundamentally there can be variance in how many and which valves and routes should be combined to create a single component. There are also differences in the way group represent their components. For example, Grimmer et. al assumes each component has four terminals located at each corner, while others assume pins can be located at other locations on a component's edge, as illustrated in Figure 6.2 [17]. The granularity of the device specification can also affect algorithmic performance, with larger granularities requiring less memory and processing time but generating less precise results. None of these are problems when generating benchmarks, per se, but without directly disseminating the results they are nearly impossible to reproduce faithfully.

Since these benchmarks have to be created by hand, they are also limited in the number of components and overall complexity. Recent work on physical design automation

has been limited to devices with at most 65 components, including layout problems formulated as boolean satisfiability (SAT) [17] and integer linear programming (ILP) [59], which will not scale. Subsystems of existing benchmarks are sometimes replicated to create a new benchmark to test the algorithm's scalability. In the case of ChIP (4IP) and ChIP (10IP), which is the same base design with 4 and 10 mixing subsystems respectively. While this does increase the scale of the benchmark, it also creates a very heterogeneous device which may be more amenable to certain classes of algorithms.

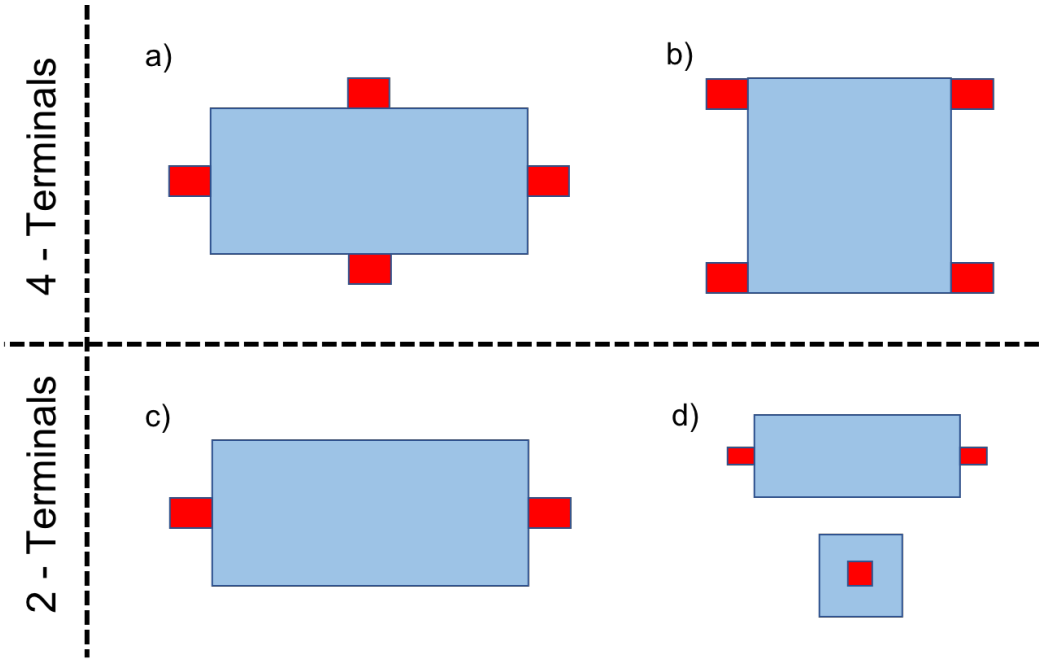


Figure 6.2: This illustrations in this figure represent how different design automation systems represent the microfluidic components they use for placement and routing. Different algorithms assume fixed positions for terminals where the channels can connect (red) on the components' boundary (blue). (a-b) Four terminals are assumed either at the corners of the bounding box or at the center of each bounding box wall [38, 17, 64] (c-d) Two terminals are assumed for each component existing at the center of two opposing bounding box edges, with the latter including a single terminal in the component's center [13, 59]. These variations in the basic component models induce variability in the layouts generated by the algorithms.

6.3 Contributions

The main contributions of this work is a standardized format for specifying continuous-flow microfluidic architectural netlists, a comprehensive set of benchmarks which aim to facilitate direct comparison of design automation algorithms, a set of metrics for quantitatively discussing these and future benchmarks, and a tool for generating new benchmarks meeting the desired complexity. Here we further describe each of these contributions.

6.3.1 Standard Interchange Format

A standard notation for describing a continuous-flow microfluidic device architecture is necessary to allow for multiple groups to be able to describe new benchmarks in a replicable manner and effectively evaluate new algorithms against existing ones. To this end, the ParchMint standard interchange format is introduced as an architectural netlist description standard. These netlists take the form of a JSON file, which has become a well-known standard in recent years for structured data and can be parsed easily in almost any programing language. To create a concrete and verifiable interchange standard, we have also provided a JSON-Schema file on the benchmark website [1] that is used to describe the fields necessary in a JSON netlist [50]. Through this, new JSON architectural netlists can be verified easily against the schema for correctness. Here we present a brief description of some of the different fields in the interchange format and their usage in current and future benchmarks.

[1] parchmint.org

The architecture netlist is primarily described through three top-level object lists: *layers*, *components*, and *connections* as shown in Figure 6.3. Every object within the system has a name and id field, which represent a human readable label and a unique id that can be referenced by other objects, respectively. The layers list can contain any number of layer objects. While the current benchmark suite and the majority of devices fabricated today contain only a flow or a flow and control layer, due to emerging methods of manufacturing there is growing interest in allowing for additional layers either to allow multi-layer routing through vias or to integrate sensors and electronic layers into the design automation process. The component objects contain a layer field representing the layers it should be present on, the x- and y-span fields which collectively describe the bounding box associated with the component and a list of ports. Each port contains an x and y location value which represents that ports location on the edge of the component bounding box and a unique label. The connection objects also contain a layer list, as well as a single source and a list of sinks. Both the source and the sink contain a component field which references the id of the source/sink component(s) that that connection routes between as well as a port field that references which port the connection should use on the specific component. Multiple sinks can be supplied to allow for multinet-style connections that start from a single source and branches to end at multiple sinks.

This standard is required, at its core, to represent the architectural netlist of a microfluidic device. To make comparisons between different placement and routing strategies easier, the standard presented here is also capable of containing placement and routing information allowing for easy external evaluation and validation. The standard has the added

benefit of allowing groups using the standard to publish their placed or routed netlists which can be used by other groups to implement new routers or post-processing steps without the need to re-implement the original algorithm. The *features* object list at the top level is used to describe the concrete placement and routing information of the abstract components and connections to support additional post-processing and design rule checking steps. This list can contain two types of objects, *component features* and *connection features*. The component features contain location information for a component, and only one component feature can exist for each component object in the component list. The connection feature contains a beginning and ending point for a straight line segment of a connection and allows multiple straight line segments that constitute a single connection object in the connection list.

As the interchange format is stored in JSON, it allows researchers to store additional information in the form of *custom* fields and *params* fields as seen in Figure 6.3. We foresee researchers utilizing this extensible format to store additional information regarding the device like emerging layout constraints, custom design rules [26], specialized component/connection physical design information, and protocol/assay descriptions [65, 71] necessary for specialized algorithms. In addition to parameters for emerging design requirements, the custom fields and params fields should also encode any values necessary for the correct processing or validation of a netlist. Any design algorithms that require external values to generate a specific result netlist should encode those values within the params field so that future designers can easily reuse and validate those results.

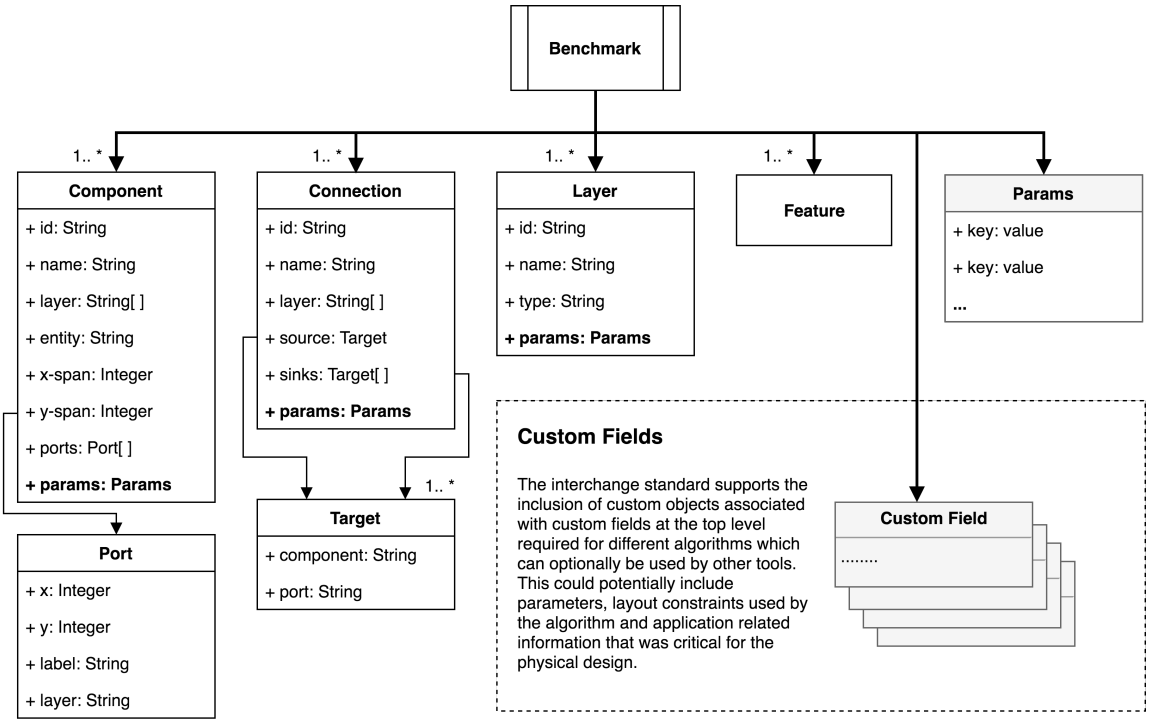


Figure 6.3: This new standard interchange format has collections of *components*, *connections* and *layers* that represent the abstract information about the benchmark. The *features* collection represents the detailed device layout once it is placed and routed by a design algorithm and finally the ability to add additional data that is specific to different design automation algorithms.

6.3.2 Comprehensive Benchmark Suite

The benchmark suite presented here has been developed to be agnostic existing physical design algorithms. We have attempted to create a representative set of benchmarks that exhibit the attributes that are commonly seen in biological experiments and shown in existing real-world designs. The majority of benchmarks currently being used in physical design automation publications consist of fewer than 100 components. As a claim of mVLSI is that you can fit thousands of valves onto a single device, however, the current benchmarks do not represent devices of this scale. Here we present benchmarks based on real-world

devices that scale up to over 400 components and present a method for generating much larger benchmarks in Section 6.3.3. All these benchmarks have been made available online [2] in the JSON interchange format and represent planar undirected multigraphs. Here we introduce each benchmark subset and briefly describe their origins:

Assay Inspired Benchmarks

These benchmarks were created by examining images from a previous microfluidic device publication for performing a complete biological assay. The netlists were created using the extracted component and connection graph from the device images and applying a general set of component descriptions across the entire architecture.

- **Chromatin Immunoprecipitation (chip):** an automated DNA-protein interaction device [66]
- **General Purpose Microfluidic Device (gpmfd):** a device for performing multiple mixing and storage operations [61]
- **Molecular Gradient Generator (mgg):** a device for generating five concentration levels of a two-sample molecular mixture [52].
- **HIV Immunoassay (hiv):** a bead-based HIV1 p24 sandwich immunoassay device [34].
- **auqaflex-3b & aquaflex-5a:** mVLSI lab-on-a-chip netlists provided by Microfluidic Innovations, LLC based on the work of Amin et. al [1]

[2] parchmint.org

Application-converted Benchmarks

These benchmarks are a set of generated architectures, **synthetic_N** (1-7), based on the DTU electrowetting-based digital microfluidic bioassays [44] and converted to netlists using an architectural synthesis method based on the one introduced by Minhass et. al [45]. Some of the netlists that were generated using this method were non-planar, which was corrected by manually removed valves and connections until the netlist was planar.

Biologically Inspired Benchmarks

This set includes benchmarks taken from Ref [26]. These designs are inspired by devices which were used in synthetic biology experiments and were reviewed by Huang et. al [27].

- **flow_focus**: a chip that can produce water droplets in a continuous flow of oil
- **grad_cells**: a chip that can separate inputs into spatial gradients
- **hasty**: a chip that uses a grid of cell traps to create a visually readable sensing array out of engineered bacteria
- **logic04**: a multi-input chip that facilitates the long-term monitoring and control of fluids between cell traps
- **multi_input**: a chip that takes multiple input fluids, mixes subsets of them, and traps cells in a central cell trap for culturing
- **net_mux**: a chip which uses pressure control lines to multiplex inputs and selectively mix chosen inputs and used in oligo-nucleotide synthesis

- **rotary16**: a rotary pump with 16 inputs and 16 outputs
- **rotary_cells**: a rotary pump which takes one input from a 4×1 mux and directs the contents to one of four cell traps
- **simple**: a chip which allows selection of one of two inputs to direct to a cell trap
- **tdroplet**: a cell trap which switches between two t-configuration droplet generators

Grid Benchmarks

This benchmark set includes designs **grid_N** (2-12) which contain generic $N \times N$ grids of cell-traps with valves that allow for the selection of either horizontal or vertical rows.

6.3.3 Scribe

Even with the *Grid* benchmarks nearing 500 components, larger benchmarks will be needed to keep up with the evolving demands of microfluidic designers and to stress test physical design automation algorithms. To address this issue, we introduce *Scribe*, an algorithm which allows the user to set desired parameters such as the number of components, average connectivity, and area utilization which will be used to generate designs exhibiting the characteristics observed in the benchmark suite or as input by the user. These designs can be generated to be arbitrarily large but with a known solution, allowing existing algorithms to be tested for scalability. Pseudocode for the Scribe algorithm is presented in Figure 6.4 with full code available on the benchmarking website [3].

[3] parchmint.org

Scribe first takes scans of all component definitions available to it from a large and expandable set and calculates the average available component size. The average available component size (C_{Avg}), the number of components (N), the target utilization (U) are then used to set the size of the device (line 3). When N components are added to the device, it will approximate the utilization U (if set). The placement loop (lines 4 – 7) places a new component c_i (from the available entity set) at a *valid* random (x, y) coordinate, where a *valid* (x, y) is any coordinate that has no component $c_j \neq c_i$ intersecting the bounding box: $([x, x + c_i.w], [y, y + c_i.h])$. The routing loop (lines 9 – 16) routes connections until the average connectivity is met ($d < D_{Avg}N$) or no new routes can be generated. The max degree D_{Max} constraint is never violated (if set). Since Scribe is capable of generating a huge range of designs, and designs of interest would likely be untenable for current methods, we have omitted reporting results for generated benchmarks.

6.3.4 Benchmark Space

ParchMint is an actively-developed benchmark suite; as the field progresses, we expect to include additional benchmarks with increased size and complexity and which represent new and emerging trends in microfluidic design. To accommodate future advances in microfluidics and their applications, it is necessary to be able to characterize the complexity of benchmarks quantitatively. This will allow the benchmark suite to evolve to best mimic the types of devices that are of interest to microfluidic designers.

Here we present some design *attributes* which were determined by qualitatively examining the structure of the benchmarks and identifying the properties associated with

Require: N: Number of components,
Target utilization U as a %,
$D : (Avg, Max)$
Ensure: Architecture A
$u \leftarrow 0$, current utilization
Let C_{avg}: Average entity size
$W, H \leftarrow C_{avg}.w * N/U, C_{avg}.h * N/U$
while $n < N$ **do**
  $A \leftarrow c_i =$new$component(x, y)$
  Place at valid(x, y)
end while
$d \leftarrow 0$
while Degree $d < D_{Avg} * N$ **do**
  Select component c_i such that $c_i.d < D_{Max}$
  if $r_i =$ Route(c_i) successful **then**
    $A \leftarrow r_i$
    $c_i.d \leftarrow c_i.d + 1$
    $d \leftarrow d + 1$
  end if
end while
return A

Figure 6.4: Scribe pseudocode for generating custom benchmarks similar to real-world benchmarks.

the various benchmark netlists. Since a microfluidic architectural netlist can be viewed as a graph, with nodes representing components and edges representing connections, it is often useful to use graph theory terms when describing netlist properties. Therefore, these properties were then converted into quantitative *attributes* that are typical for graph networks, and categorized into either *component attributes* or *connection attributes*.

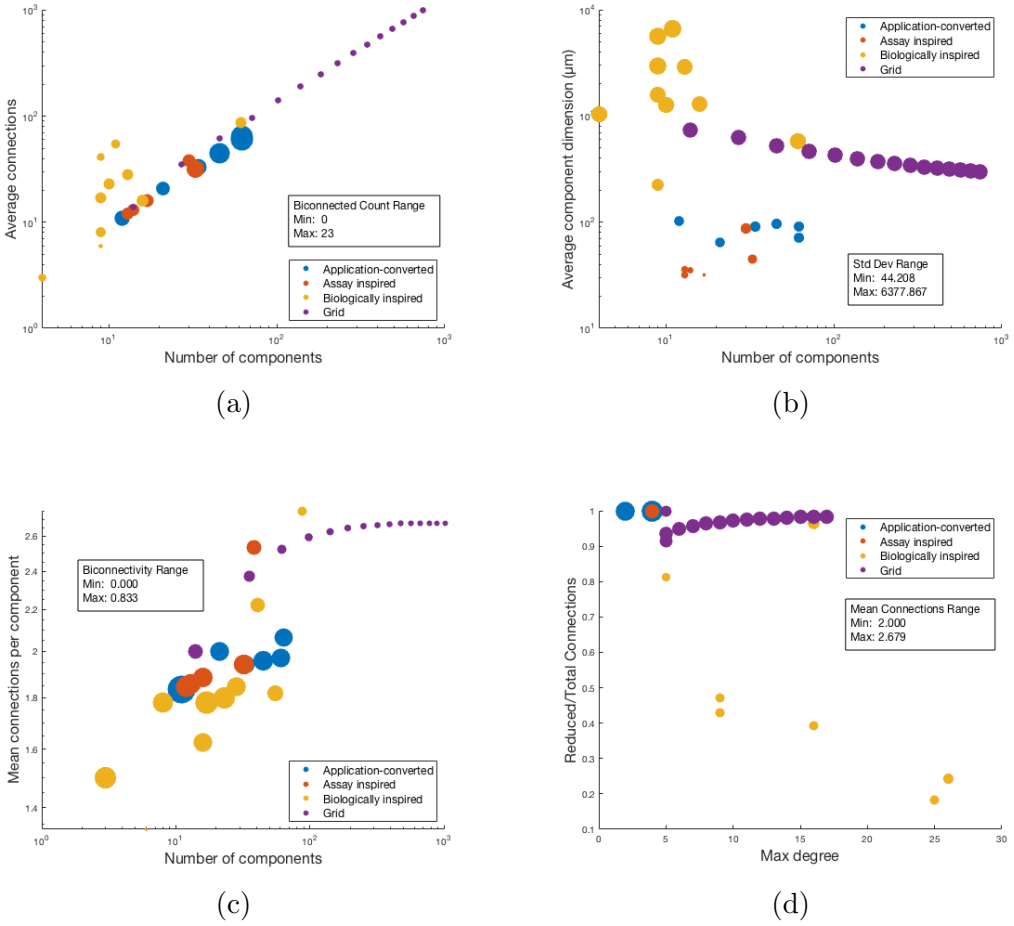


Figure 6.5: The above graphs visualize various component/connection attributes, heuristics (marker radius) of all the benchmark sets. (a) Helps visualize the variation in the connection complexity and the scale. The figure also shows how most of the assay inspired and application converted benchmarks occupy the same complexity space. (b) Helps visualize the variation in physical dimensions of the components. As it can be seen the *Biologically Inspired* and the *Grid* benchmarks have far larger components with large variations in the dimensions of the components. (c) Shows how the variation in the connection density per component. (d) Shows how the connection complexity show in (a) and (c) can be effectively reduced when we look at the netlist connectivity. We can see that the benchmarks with the highest connections to a component can reduced drastically indicating buses.

Component Attributes

- **Total Components:** the total number of components that are present in the netlist.

  It is one of the main metrics for describing benchmark complexity.

- **Average/Minimum/Maximum Component Area:** the average/minimum/maximum area take up by (a) component(s) in the netlist. It shows the variance between components, and if the design has homogeneous or heterogeneous component sizes.
- **Total Biconnected:** the number of components in the system that have exactly two connections. Biconnected components can imply a pipeline stage or connection between device subsystems that are relatively independent.

Connection Attributes

- **Total Connections:** the total number of connections that are present in the netlist. This combined with the number of components is currently the largest factors when considering benchmark complexity.
- **Average/Minimum/Maximum Connectivity:** equivalent to the vertex degree in graph theory, this is the average/minimum/maximum number of channels connected to a component in the netlist. It illustrates whether connections are distributed through the system or aggregated in a small number of components.
- **Reduced Connections:** the number of connections, reduced by treating groups of channels that connect the same two components as a single bus. This value would be much lower than the total number of connections in a device with components that are highly interconnected, suggesting that some of the complexity can be reduced effectively.

These benchmark characterization *attributes* will give researchers a common vocabulary to quantify where their algorithms succeed and where they are limited. It will also

allow for the creation and comparison of new benchmarks that stress on different combinations of *attributes* in the continuous-flow microfluidic benchmark space. For instance, the benchmarks presented here have a wide range in the number of components, average connectivity, biconnectivity, and average component dimensions as illustrated in Figure 6.5a, Figure 6.5b, Figure 6.5c. However, there is lower variance in maximum component degree, and reduced connections as illustrated in Figure 6.5d. These metrics provide a quantitative means for the targeted creation of new benchmarks in the future that are substantively different from the current set.

6.4 Case Study

6.4.1 Physical Design Flow

Current continuous-flow microfluidic devices can be viewed logically as having two layers; a flow layer with components and channels through which the reagents used in the experiment move, and the control layer which is used to actuate the valves and manipulate the movement of fluids. The device shown in Figure 6.1 is an example of one such microfluidic device that was intended for high throughput drug screening. The two logical layers, flow, and control is colored in blue (the flow layer colored in blue and green in the polydimethylsiloxane (PDMS) device) and red respectively in the figure. The majority of current physical design automation tools and algorithms automate the placement and routing of the flow layer, control layer, or the co-design of both. The benchmarks presented here were all processed using two different algorithms, each of which is outlined here:

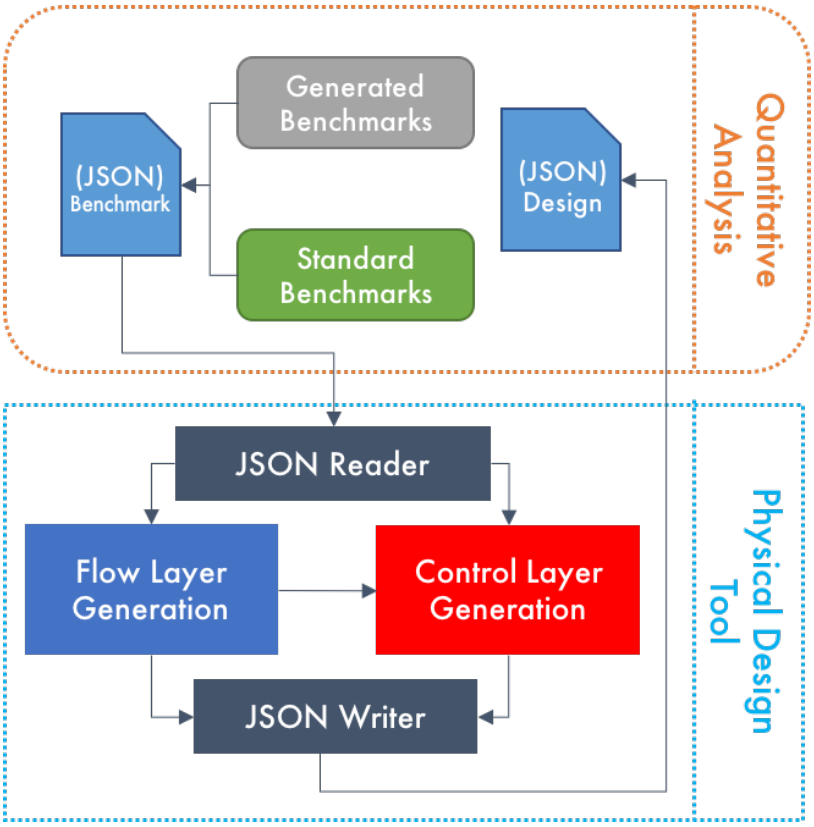


Figure 6.6: Both the benchmarks presented here and those generated by Scribe use the JSON standard interchange format, which can be easily read and parsed for metric information and quantitative analysis. Individual teams systems can read this format, and the data processed through flow layer and control layer generation allowing the researcher to capture and share the information at any step. This makes it an excellent format for not only describing benchmarks but describing the current state of a partially processed microfluidic design.

Flow Layer Generation

The first step in the physical design process is to take the components in the netlist and to lay them out on a 2D plane with no two component bounding boxes overlapping. McDaniel et. al [41] and Huang [26] used SA to place the components and evaluated different objective functions to optimize. BU has implemented a SA method based on these works to evaluate this class of algorithms. It should be noted this method is differs from the SA method

introduced in Chapter 2. Additionally, we use the Planar Placement method introduced in Chapter 2 to evaluate these *Planar Embedding* classes of algorithms.

Next, we need to route the connections between components. Since these channels carry fluid, any intersections would cause the fluids they contain to contaminate each other, thus nullifying the results of the experiment and necessitates the planarity requirement. Huang [26] implemented a routing method based on Hadlock's algorithm which increases the cost of creating intersections to avoid them. BU implemented a similar *Hadlocks* method and paired it with SA for the presented results. The Network-flow based routing approach introduced in Chapter 2 to route the flow layer which made multiple rip, re-order and re-route attempts if routes intersected is paired with Planar Embedding for the presented results. Both methods mark the routing as a failure if intersections are found.

Control Layer Generation

Previous algorithms have been designed for control-flow co-design [59], control optimization [46], and control routing [71]. Control layer generation is performed by routing each valve on the device to its pressure source pin. However, to perform control optimization and allow valves to share pressure source pins, an application file must be mapped onto the architecture. The reason being that valves can only share a pressure source when they share an actuation sequence, and this can be determined only from the application. Since we are not addressing the problem of application mapping here, not providing applications to be mapped, and since all benchmarks do not contain control layers we have chosen to exclude methods that perform control optimization [59] or base the layout on the control layer [71] to generate the physical design to avoid creating an unfair comparison.

Fabrication

The last step in any design automation framework is the fabrication. The majority of microfluidic devices in literature are currently fabricated in PDMS or glass using soft-lithography [51], a process where a photo-resist and mask are applied to a substrate and developed to create inverse channels. Additionally, multiple valve technologies have been developed each of which requires different parameters based on their type, desired actuation pressure, and substrate material [62, 19]. Design parameters for fabrication are out of scope for this work because of the significant variance between technologies, materials, and individual fabricators.

All of the benchmarks mentioned in Table 6.1 were designed to target soft lithography as a manufacturing method. However, the majority of the benchmarks have not been fabricated. Since this manufacturing process does not allow the device to have vertical *vias* between layers, all the devices manufactured using this process are required to be planar. This fundamental feature of microfluidic device architectures has been exploited in the design of various algorithms used in literature. This does not reflect newer manufacturing trends such as computer numeric controlled (CNC) milling [32], 3D printing [16], and laser cutting [63]. These methods have support for the fabrication of *vias* to varying degrees and introduce the viability of non-planar architectures. While both the Hadlocks and Network-flow routing algorithms target a planar design, the benchmarks list of layers allows for additional layers and vias between them to be introduced in future algorithms.

6.4.2 Metrics

There are several standard metrics currently used to evaluate the effectiveness of a physical design algorithm. *Total area* is a primary metric for reporting the total square area that the device consumes. *Utilized area* measures the amount of the total space that is used by the component and connection and dividing that by the total area to show it as a percentage. *Total channel length* measures the total length of all channels for all connections in the system and *average channel length*, therefore, is the total channel length divided by the number of connections.

6.5 Results

The flow layers of the benchmarks presented here were run through two distinct physical design automation frameworks, and the metrics from Section 6.4.2 are presented in Tables 6.2 and 6.3. Here we discuss some of the issues that were found when performing this work which should help to illustrate why a benchmark suite with a high amount of scale and feature variance is necessary to evaluate current and future design automation algorithms properly.

In all cases, both systems were capable of generating valid placements, but those placements were not always capable of yielding a valid routing. The SA and the Hadlocks system was unable to find routable layouts for the benchmark sets *Assay Inspired* and *Application-converted* even when the number of iterations was double of that used to place and route *Biologically Inspired*.

The Planar Embedding and Network-Flow algorithm only failed on the *net_mux* benchmark in the *Biologically Inspired Benchmarks* set. *net_mux* contains two components with very high connectivity, 19 connections in each case as seen in Figure 6.5d. The presence of a large number of connections to a single component created areas of high routing density which became difficult to solve. The problem is exasperated in the case of planar embedding since it does not consider a rotation step while performing placement. Since one of the high connectivity components only has the majority of its ports on one of its sides, if that component's ports faced away from the majority of its connected components then it will become even more difficult to find a valid routing.

The Planar Embedding and Network-flow method are only capable of routing the first benchmark, *grid_2*, of the *Grid Benchmarks* suite. Once again, the Planar Embedding algorithm was capable of creating a valid placement for the components; however, that placement is inferior at supporting a grid-like architecture structure. The reason for this lies with the underlying embedding method that is used to create the initial placement. The Chrobak-Payne straight line planar embedding algorithm [8] functions by using triangulation to place its points within a 2D plane. With the *grid* Benchmarks, it tends to place them along the diagonal of the plane and within sub-triangles within the system. This arrangement leads to congestion when routing along/across the diagonal and a high rate of routing failure. The *grid_2* benchmark is the largest benchmark that can be routed before the congestion is high enough to cause routing failures in this type of layout.

While the Planar Embedding algorithm was successful in generating the layout for all of the *Assay Inspired* and the *Application Converted* benchmarks it was observed that the

SA and Hadlocks algorithm performed very poorly on these benchmark sets. The simulated annealing algorithm [26] minimizes a cost function that based on total connection length and the total area of the chip. After extensive testing, it was found that the algorithm often optimized for the most compact layout which was unroutable (non-planar). The combination of factors where (1) the small component sizes (Figure 6.5b) (2) the sparse connectivity (Figure 6.5a) 3) non-inclusion of routability as a factor in the objective function consistently pushed the algorithm to generate non-planar layouts.

Some amount of difference in benchmark metrics stems from the amount of buffer space that is reserved around components and connections for routing. This difference is observed in the *simple* and *flow_focus* benchmarks from the *Biologically Inspired Benchmarks* set, where the relatively large amount of buffer space reserved in the Hadlocks routing method coupled with the small component sizes means that there is a significant amount of wasted space. On the other hand, the Network-flow method reserves a relatively small amount of buffer space but may be required to make more rip, re-order, and re-route iterations before it can find a valid route. These buffer spaces reserved for routing can also make the exchange between teams of the standard interchange format between physical design steps difficult, as placements using one method may not meet the buffer routing requirements of another. This type of variability between methods necessitates the inclusion of the params field within the standard interchange format.

Since both the routers evaluated here are grid-based routers, the memory constraints can become a critical issue. Benchmarks with placements that use a large area and utilize a high precision required more memory than what was available to build a full

routing grid. Both the Hadlocks router and the Network-flow router can utilize a 1μm, 10μm, or 100μm precision routing grid through a pre-processing method that scales the entire architecture netlist down by a factor before the placement step. The scaling decreases the accuracy of the results by a small amount but allows for the processing of much larger benchmarks. Many of the benchmarks presented here were processed with a 10μm routing grid, including those processed with the Hadlocks router, and are annotated in Tables 6.2 and 6.3.

Additionally, issues arose during the routing phase because of the different methods that each router used to determining a starting and ending port. While some components can function with its connections routed to arbitrary ports, others require specific ports to be utilized by a specific connection. The Network-flow algorithm assumes that it can perform its port assignments, while the Hadlocks router requires all ports to be pre-assigned before it can perform routing. Because of this discrepancy, ports were assigned to all connections in the benchmarks, even when the component may not require a specific port assignment.

6.6 Conclusion

ParchMint started as an attempt to standardize benchmarks individually developed at BU CIDAR lab and UCR to enable more robust testing of the two institutions' respective algorithmic toolchains. This effort was subsequently expanded to create the benchmark suite that we present here, which represents the beginning of a longer-term project to establish a more standard and sophisticated nomenclature to specify microfluidic device architectures and design algorithms. In the future, we hope to incorporate input from domain experts

and design researchers to complement current efforts, which were primarily carried out by experts in design automation. We hope to ensure that the benchmark suite continues to grow in size and that it incorporates that latest advances in microfluidic techniques and technology. We expect this effort to encompass all levels of the microfluidic design toolchain, including similar standards, benchmarks, and analysis for component entities, control systems, and device functionality and hope that other groups will join in this ongoing effort.

Table 6.2: ParchMint benchmark results for the Planar Embedding & Network-flow method

Benchmark				Planar Embedding & Network-flow			
				Area		Route Length	
Name	NV	NE	DIM	Size (WxH)	%	Total	Avg.
chip	33	32	6318	2181x2181	4.8	19933	622.91
gpmfd	13	12	3469	701x701	9.93	3695	307.92
mgg	30	38	16090	3258x3258	5.02	50305	1323.82
hiv	13	12	3700	761x761	8.9	3416	284.67
aquaflex-3b	14	13	3464	802x802	8.24	4489	345.31
aquaflex-5a	17	16	2905	915x915	6.69	6606	412.88
synthetic_1	21	21	8409	1699x1899	6.21	13449	640.43
synthetic_2	12	11	15483	1410x1610	8.33	3306	300.55
synthetic_3	34	33	15067	3702x3902	3.73	26963	817.06
synthetic_4	34	33	15058	3692x3892	3.81	35074	1062.85
synthetic_5	46	45	15869	5224x5524	2.81	81353	1807.84
synthetic_6	62	64	11611	5700x5850	2.31	50512	789.25
synthetic_7	62	61	14727	6720x7170	2.34	212130	3477.54
flow_focus	4	3	7220000	3252x5172	85.88	5876	1958.67
grad_cells	9	17	13697658	28870x1525†	24.36†	129980†	18568.57†
hasty	10	23	4418032	12934x12258	27.9	50253	5583.67
logic04	9	41	13697658	15860x38740†	32.39†	49260†	9852.0†
multi_input	16	16	2663087	22600x18890†	10.12†	85490†	9498.89†
net_mux	61	87	1297976	#	#	#	#
rotary16	11	55	36640000	81630x20470†	19.49†	50910†	12727.5†
rotary_cells	13	28	8915763	21180x20250†	14.35†	42800†	7133.33†
simple	9	6	55000	1063x1513	22.14	1156	289
tdroplet	9	8	4316044	14030x15750†	17.76†	48950†	8158.30†
grid_2	14	14	1283328	3320x4512	18.36	22982	2298.2
grid_3	27	35	1146513	§	§	§	§
grid_4	46	62	924875	§	§	§	§
grid_5	71	97	851863	§	§	§	§
grid_6	102	140	726669	§	§	§	§
grid_7	139	191	633513	§	§	§	§
grid_8	182	250	562067	§	§	§	§
grid_9	231	317	557066	§	§	§	§
grid_10	286	392	506888	§	§	§	§
grid_11	347	475	465583	§	§	§	§
grid_12	414	566	431027	§	§	§	§

Key	# : High Density Component † : Scaled by 1/10 § : Poor Embedding
	NV : # Components NE : # Connections DIM : Avg. Area % : Utilization

Table 6.3: ParchMint benchmark results for the Simulated Annealing & Hadlocks method

Benchmark				Simulated Annealing & Hadlocks †			
				Area		Route Length	
Name	NV	NE	DIM	Size (WxH)	%	Total	Avg.
chip	33	32	6318	*	*	*	*
gpmfd	13	12	3469	*	*	*	*
mgg	30	38	16090	*	*	*	*
hiv	13	12	3700	*	*	*	*
aquaflex-3b	14	13	3464	*	*	*	*
aquaflex-5a	17	16	2905	*	*	*	*
synthetic_1	21	21	8409	*	*	*	*
synthetic_2	12	11	15483	*	*	*	*
synthetic_3	34	33	15067	*	*	*	*
synthetic_4	34	33	15058	*	*	*	*
synthetic_5	46	45	15869	*	*	*	*
synthetic_6	62	64	11611	*	*	*	*
synthetic_7	62	61	14727	*	*	*	*
flow_focus	4	3	7220000	5200x6620	41.95	2580	860
grad_cells	9	17	13697658	36240x8500	34.73	26790	1786
hasty	10	23	4418032	20690x6000	36.90	48990	2041.25
logic04	9	41	13697658	26080x20400	38.63	75120	5365.71
multi_input	16	16	2663087	22320x10270	18.76	34580	2161.25
net_mux	61	87	1297976	17470x22440	20.32	95020	1727.64
rotary16	11	55	36640000	29320x25980	43.79	349020	7932.27
rotary_cells	13	28	8915763	30480x11340	18.13	32120	1889.41
simple	9	6	55000	8060x6370	0.67	177750	22218.75
tdroplet	9	8	4316044	12670x14040	22.08	16140	1614
grid_2	14	14	1283328	7500x11410	3.70	10050	1005
grid_3	27	35	1146513	8700x16740	6.35	20980	1049
grid_4	46	62	924875	9900x15780	8.72	31210	917.94
grid_5	71	97	851863	11100x20950	10.61	50650	974.04
grid_6	102	140	726669	12300x22400	11.23	98930	1236.63
grid_7	139	191	633513	13500x23750	11.67	125290	1160.09
grid_8	182	250	562067	15410x25340	11.28	136890	1053
grid_9	231	317	557066	15930x28100	14.18	474210	1992.48
grid_10	286	392	506888	17110x27240	15.45	342090	1413.60
grid_11	347	475	465583	18310x31060	14.19	404210	1255.31
grid_12	414	566	431027	19520x32630	14.0	460050	1185.0

Key	* : Illegal Layout			
	NV : # Components	NE : # Connections	DIM : Avg. Area	% : Utilization

Chapter 7

Conclusions

Here we've presented a number of different techniques that make up the placement, routing, and post-processing core of a microfluidic design toolchain for passive devices. While the primary focus of this work was on the flow layer, all the techniques presented here are amenable to the inclusion of a control layer with the post-processing techniques considering it explicitly.

We began with tools for the automated placement and routing of the flow layer. The planar-embedding based Planar Placement methods, made up primarily of the baseline expansion and Diagonal Component Expansion (DICE) methods presented in Chapter 2, create layouts that are highly routable in practice over a number of scales and topologies as illustrated in Chapter 6. With these placement methods we also introduce a Network-flow based routing method in Chapter 2, which is highly capable of finding a set of valid routes for a number of different placement algorithms while not introducing any additional intersections. This planar routing method is important since the introduction of any in-

tersections, requiring the insertion of a switch to correct, makes it impossible for use in designing passive devices. We then present directed placement in Chapter 3, which generate much better designs similar to those created by domain experts, but tuned to a smaller subset of topologies.

In addition to these automated design methods, we also introduce the first post processing algorithms for microfluidic devices that are designed to process both automated methods as well as designed created by domain experts. The seam carving post processing method introduced in Chapter 4 is capable of finding area and channel connection reductions in non-optimal device designs while considering both the flow and control layers and without the intervention of the device designer. The automated arraying method in Chapter 5 allows designers to take a previously designed and tested device and array a subsection of it. This can be used to increase a devices throughput, created redundancy for a process, or add fluidic memory. This is accomplished through a simple area selection by the designer, automating a time consuming and error prone task without circumventing their expertise in designing the original device. It is our hope that these algorithms will represent the beginning of a movement toward design acceleration, the idea that algorithms should help augment steps of the design process in addition to automated them.

Finally, we introduce the benchmarking suite ParchMint in Chapter 6. This work represents the first step in a long-term project between University of California Riverside (UCR) and Boston University (BU) that was developed through discussions with other design automation researchers, tool designers, and microfluidic device designers. Through these interactions, we came to the conclusion that the current state of microfluidic design

automation and acceleration analysis needed to be improved in order to better evaluate if the algorithms being created were capable of creating valid device designs over the entirety of the highly diverse and scalable world of continuous-flow microfluidic devices. The standard, benchmark netlists, generators, benchmark and result metrics, and algorithmic results we present are the first step in trying to improve the discussion around design automation and acceleration algorithms in this space. We hope to expand this benchmark suite going forward to include similar benchmarks and analysis for component entities, control systems, and device functionality. Our hope is that this work into benchmarks and standards will help accelerate the design automation and acceleration space to move beyond theoretical discussions about device layouts and into fabrication and device execution as validation steps.

The field of continuous-flow microfluidic design automation and acceleration is a rapidly evolving one. New fabrication techniques, device components, surface treatments, and external systems are constantly being developed. This changes not only the capabilities of microfluidic devices, but also the requirements that need to be considered when creating tools to support designers. This constant evolutions makes the process of creating new algorithms, methods, and tools a moving target. Because of this, design automation developers will increasingly need to work directly with device designers and domain experts to better understand their needs and be able adapt to changing technologies.

Bibliography

[1] Ahmed M. Amin, Raviraj Thakur, Seth Madren, Han Sheng Chuang, Mithuna Thottethodi, T. N. Vijaykumar, Steven T. Wereley, and Stephen C. Jacobson. Software-programmable continuous-flow multi-purpose lab-on-a-chip. *Microfluidics and Nanofluidics*, 15(5):647–659, 2013.

[2] Emre Araci. Stanford Microfluidics Foundry, 2005.

[3] Ismail Emre Araci and Stephen R Quake. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. *Lab on a Chip*, 12(16):2803–2806, 2012.

[4] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics*, 26(3):10, 2007.

[5] Frederick K Balagaddé. Long-Term Monitoring of Bacteria Undergoing Programmed Population Control in a Microchemostat Long-Term Monitoring of Bacteria Undergoing Programmed Population Control in a Microchemostat. *Science*, 137(2005):137–140, 2014.

[6] John M Boyer and Wendy J Myrvold. On the Cutting Edge: Simplified O(n) Planarity by Edge Addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[7] H. Nelson Brady. An Approach to Topological Pin Assignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(3):250–255, 1984.

[8] M. Chrobak and T. H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995.

[9] Brian Crites, Karen Kong, and Philip Brisk. Diagonal Component Expansion for Flow-Layer Placement of Flow-Based Microfluidic Biochips. *ACM Transactions of Embedded Computer Systems*, 16(18), 2017.

[10] Brian Crites, Karen Kong, and Philip Brisk. Reducing Microfluidic Very Large Scale Integration (mVLSI) Chip Area by Seam Carving. *In Proceedings of the Great Lakes Symposium on VLSI*, pages 459–462, 2018.

[11] Weiming Dong, Ning Zhou, Jean-Claude Paul, and Xiaopeng Zhang. Optimized image resizing using seam carving and scaling. *ACM Transactions on Graphics*, 28(5):1, 2009.

[12] Jamil El-Ali, Peter K Sorger, and Klavs F Jensen. Cells on chips. *Nature*, 442(7101):403–411, 2006.

[13] Morten Chabert Eskesen, Paul Pop, and Seetal Potluri. Architecture Synthesis for Cost-Constrained. *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, (3):618–623, 2016.

[14] D. Falconnet, A. Niemistö, R. J. Taylor, M. Ricicova, T. Galitski, I. Shmulevich, and C. L. Hansen. High-throughput tracking of single yeast cells in a microfluidic imaging matrix. *Lab on a Chip*, 11(3):466–473, 2011.

[15] Cong Fang, Yanju Wang, Nam T. Vu, Wei Yu Lin, Yao Te Hsieh, Liudmilla Rubbi, Michael E. Phelps, Markus Müschen, Yong Mi Kim, Arion F. Chatziioannou, Hsian Rong Tseng, and Thomas G. Graeber. Integrated microfluidic and imaging platform for a kinase activity radioassay to analyze minute patient cancer samples. *Cancer Research*, 70(21):8299–8308, 2010.

[16] Hua Gong, Adam T. Woolley, and Gregory P. Nordin. High density 3D printed microfluidic valves, pumps, and multiplexers. *Lab on a Chip*, 16(13):2450–2458, 2016.

[17] Andreas Grimmer, Qin Wang, Hailong Yao, Tsung Yi Ho, and Robert Wille. Close-to-optimal placement and routing for continuous-flow microfluidic biochips. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pages 530–535, 2017.

[18] William H. Grover, Robin H. C. Ivester, Erik C. Jensen, and Richard A. Mathies. Development and multiplexed control of latching pneumatic valves using microfluidic logical structures. *Lab on a Chip*, 6(5):623, 2006.

[19] William H. Grover, Alison M. Skelley, Chung N. Liu, Eric T. Lagally, and Richard A. Mathies. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. *Sensors and Actuators, B: Chemical*, 89(3):315–323, 4 2003.

[20] Carl L Hansen, Morten O A Sommer, and Stephen R. Quake. Systematic investigation of protein phase behavior with a microfluidic formulator. *Proceedings of the National Academy of Sciences of the United States of America*, 101(40):14431–6, 2004.

[21] Akihiro Hashimoto and James Stevens. Wire Routing by Optimizing Channel Assignment within Large Apertures. *DAC '71 Proceedings of the 8th Design Automation Workshop*, pages 155–169, 1971.

[22] Wajid Hassan. Microfluidic flow-based biochips benchmark suite, 2012.

[23] Jong Wook Hong and Stephen R. Quake. Integrated nanoliter systems, 2003.

[24] Jong Wook Hong, Vincent Studer, Giao Hang, W. French Anderson, and Stephen R. Quake. A nanoliter-scale nucleic acid processor with parallel architecture. *Nature Biotechnology*, 22(4):435–439, 2004.

[25] Kai Hu, Trung Anh Dinh, Tsung-Yi Ho, and Krishnendu Chakrabarty. Control-Layer Optimization for Flow- Based mVLSI Microfluidic Biochips. In *Cases*, number June, pages 1–9, 2014.

[26] Haiyao Huang. *Fluigi: An End-toEnd Software Workflow for Microfluidic Design*. PhD thesis, 2016.

[27] Haiyao Huang and Douglas Densmore. Integration of microfluidics into the synthetic biology design flow. *Lab on a Chip*, 14(18):3459–3474, 2014.

[28] Wei-Lun Huang, Ankur Gupta, Sudip Roy, Tsung-Yi Ho, and Paul Pop. Fast Architecture-Level Synthesis of Fault-Tolerant Flow-Based Microfluidic Biochips. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1671–1676, 2017.

[29] Paul J. Hung, Philip J. Lee, Poorya Sabounchi, Robert Lin, and Luke P. Lee. Continuous perfusion microfluidic cell culture array for high-throughput cell-based assays. *Biotechnology and Bioengineering*, 89(1):1–8, 2005.

[30] Jungkyu Kim, Erik C. Jensen, Mischa Megens, Bernhard Boser, and Richard A. Mathies. Integrated microfluidic bioprocessor for solid phase capture immunoassays. *Lab on a Chip*, 11(18):3106–3112, 2011.

[31] Casimir Kuratowski. Sur le problème des courbes gauches en Topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[32] Ali Lashkaripour, Ryan Silva, and Douglas Densmore. Desktop micromilled microfluidics. *Microfluidics and Nanofluidics*, 22(3):1–13, 2018.

[33] C. Y. Lee. An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.

[34] Baichen Li, Lin Li, Allan Guan, Quan Dong, Kangcheng Ruan, Ronggui Hu, and Zhenyu Li. A Smartphone Controlled Handheld Microfluidic Liquid Handling System. *Lab on a Chip*, 14(20):4085–92, 8 2014.

[35] Chun-xun Lin, Chih-hung Liu, I-che Chen, D T Lee, and Tsung-yi Ho. An Efficient Bi-criteria Flow Channel Routing Algorithm For Flow-based Microfluidic Biochips. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, pages 1–6, 2014.

[36] Joshua S. Marcus, W. French Anderson, and Stephen R. Quake. Microfluidic single-cell mRNA isolation and analysis. *Analytical Chemistry*, 78(9):3084–3089, 2006.

[37] Jeffrey McDaniel, Auralila Baez, Brian Crites, Aditya Tammewar, and Philip Brisk. Design and verification tools for continuous fluid flow-based microfluidic devices. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pages 219–224, 2013.

[38] Jeffrey McDaniel, Brian Crites, Philip Brisk, and William H. Grover. Flow-layer physical design for microchips based on monolithic membrane valves. *IEEE Design and Test*, 2015.

[39] Jeffrey McDaniel, Christopher Curtis, and Philip Brisk. Automatic synthesis of microfluidic large scale integration chips from a domain-specific language. In *2013 IEEE Biomedical Circuits and Systems Conference, BioCAS 2013*, pages 101–104. IEEE, 10 2013.

[40] Jeffrey McDaniel, William H. Grover, and Philip Brisk. The case for semi-automated design of microfluidic very large scale integration (mVLSI) chips. *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, pages 1793–1798, 2017.

[41] Jeffrey McDaniel, Brendon Parker, and Philip Brisk. Simulated annealing-based placement for microfluidic large scale integration (mLSI) chips. *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, 4(d):1–6, 2014.

[42] Carver Mead and Lynn Conway. Introduction to VLSI Systems. *IEE Proceedings I Solid State and Electron Devices*, 128(1):18, 1981.

[43] Jessica Melin and Stephen R Quake. Microfluidic Large-Scale Integration: The Evolution of Design Rules for Biological Automation. *Annual Review of Biophysics and Biomolecular Structure*, 36(1):213–231, 2007.

[44] Wajid Hassan Minhass, Paul Pop, and Jan Madsen. System-level modeling and synthesis of flow-based microfluidic biochips. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems - CASES '11*, page 225, New York, New York, USA, 2011. ACM Press.

[45] Wajid Hassan Minhass, Paul Pop, Jan Madsen, and Felician Stefan Blaga. Architectural synthesis of flow-based microfluidic large-scale integration biochips. *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems - CASES '12*, page 181, 2012.

[46] Wajid Hassan Minhass, Paul Pop, Jan Madsen, and Tsung Yi Ho. Control synthesis for the flow-based microfluidic large-scale integration biochips. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 1:205–212, 2013.

[47] H. Niederholtmeyer, V. Stepanova, and S. J. Maerkl. Implementation of cell-free biological networks at steady state. *Proceedings of the National Academy of Sciences*, 110(40):15985–15990, 2013.

[48] P. E. Gaillardon, L. Amaru, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli. The Programmable Logic-in-Memory (PLiM) computer. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (March 2016):427–432, 2016.

[49] Nicole Pamme. Continuous flow separations in microfluidic devices. *Lab on a chip*, 7(12):1644–59, 2007.

[50] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martn Ugarte, and Domagoj Vrgoč. Foundations of JSON Schema. *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, pages 263–273, 2016.

[51] Dong Qin, Younan Xia, and George M. Whitesides. Soft lithography for micro- and nanoscale patterning. *Nature Protocols*, 5(3):491–502, 2010.

[52] Minsoung Rhee and Mark a Burns. Microfluidic assembly blocks. *Lab on a chip*, 8(8):1365–73, 8 2008.

[53] Arnold Sommerfeld. Ein beitrag zur hydrodynamischen erklaerung der turbulenten fluessigkeitsbewegungen. *Atti del*, 4:116–124, 1908.

[54] Fei Su and Krishnendu Chakrabarty. Benchmarks for Digital Microfluidic Biochip Design and Synthesis, 2006.

[55] Todd Thorsen, Sebastian J. Maerkl, and Stephen R. Quake. Microfluidic Large-Scale Integration. *Science*, 298(5593):580–584, 2002.

[56] Kai-Han Tseng, Sheng-Chi You, Jhe-Yu Liou, and Tsung-Yi Ho. A top-down synthesis methodology for flow-based microfluidic biochips considering valve-switching minimization. *Proceedings of the 2013 ACM international symposium on International symposium on physical design - ISPD '13*, page 123, 2013.

[57] Tsun Ming Tseng, Bing Li, Mengchu Li, Tsung Yi Ho, and Ulf Schlichtmann. Reliability-Aware Synthesis With Dynamic Device Mapping and Fluid Routing for Flow-Based Microfluidic Biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):1981–1994, 2016.

[58] Tsun-Ming Tseng, Mengchu Li, Daniel Nestor Freitas, Travis McAuley, Bing Li, Tsung-Yi Ho, Ismail Emre Araci, and Ulf Schlichtmann. Columba 2.0: A Co-Layout Synthesis Tool for Continuous-Flow Microfluidic Biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 0070(c):1–1, 2017.

[59] Tsun-Ming Tseng, Mengchu Li, Bing Li, Tsung-Yi Ho, and Ulf Schlichtmann. Columba: Co-Layout Synthesis for Continuous-Flow Microfluidic Biochips Tsun-Ming. *Design Automation Conference*, pages 1–6, 2016.

[60] Marc A. Unger, Hou Pu Chou, Todd Thorsen, Axel Scherer, and Stephen R. Quake. Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science*, 288(5463):113–116, 2000.

[61] John Paul Urbanski, William Thies, Christopher Rhodes, Saman Amarasinghe, and Todd Thorsen. Digital microfluidics using soft lithography. *Lab Chip*, 6(1):96–104, 1 2006.

[62] H. T G van Lintel, F. C M van De Pol, and S Bouwstra. A piezoelectric micropump based on micromachining of silicon. *Sensors and Actuators*, 15(2):153–167, 1988.

[63] David I. Walsh, David S. Kong, Shashi K. Murthy, and Peter A. Carr. Enabling Microfluidics: from Clean Rooms to Makerspaces. *Trends in Biotechnology*, 35(5):383–392, 2017.

[64] Qin Wang, Yizhong Ru, Hailong Yao, Tsung Yi Ho, and Yici Cai. Sequence-pair-based placement and routing for flow-based microfluidic biochips. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, volume 25-28-Janu, pages 587–592, 2016.

[65] Qin Wang, Hao Zou, Hailong Yao, Tsung Yi Ho, Robert Wille, and Yici Cai. Physical Co-Design of Flow and Control Layers for Flow-Based Microfluidic Biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(6):1157–1170, 2018.

[66] Angela R Wu, Joseph B Hiatt, Rong Lu, Joanne L Attema, Neethan a Lobo, Irving L Weissman, Michael F Clarke, and Stephen R. Quake. Automated microfluidic chromatin immunoprecipitation from 2,000 cells. *Lab on a chip*, 9(10):1365–70, 5 2009.

[67] Angela R. Wu, Tiara L.A. Kawahara, Nicole A. Rapicavoli, Jan van Riggelen, Emelyn H. Shroff, Liwen Xu, Dean W. Felsher, Howard Y. Chang, and Stephen R. Quake. High throughput automated chromatin immunoprecipitation as a platform for drug screening and antibody validation. *Lab on a Chip*, 12(12):2190, 2012.

[68] Y Xia and G M Whitesides. Soft Lithography. *Annual Reviews in Materials Science*, 28(1):153–184, 1998.

[69] Hua Xiang, Xiaoping Tang, and Martin D F Wong. Min-cost flow-based algorithm for simultaneous pin assignment and routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7):870–878, 7 2003.

[70] Paul Yager, Thayne Edwards, Elain Fu, Kristen Helton, Kjell Nelson, Milton R. Tam, and Bernhard H. Weigl. Microfluidic diagnostic technologies for global public health. *Nature*, 442(7101):412–418, 2006.

[71] Hailong Yao, Tsung-Yi Ho, and Yici Cai. PACOR: Practical Control-Layer Routing Flow with Length-Matching Constraint for Flow-Based Microfluidic Biochips. *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pages 1–6, 2015.