Diagonal Component Expansion for Flow-Layer Placement of Flow-Based Microfluidic Biochips

BRIAN CRITES, KAREN KONG, and PHILIP BRISK, University of California, Riverside

Continuous flow-based microfluidic devices have seen a huge increase in interest because of their ability to automate and miniaturize biochemistry and biological processes, as well as their promise of creating a programmable platform for chemical and biological experimentation. The major hurdle in the adoption of these types of devices is in the design, which is largely done by hand using tools such as AutoCAD or SolidWorks, which require immense domain knowledge and are hard to scale. This paper investigates the problem of automated physical design for continuous flow-based microfluidic very large scale integration (mVLSI) biochips, starting from a netlist specification of the flow layer. After an initial planar graph embedding, vertices in the netlist are expanded into two-dimensional components, followed by fluid channel routing. A new heuristic, DIagonal Component Expansion (DICE) is introduced for the component expansion step. Compared to a baseline expansion method, DICE improves area utilization by a factor of 8.90x and reduces average fluid routing channel length by 47.4%.

CCS Concepts: • **Hardware** → **Placement**; **Emerging architectures**; *Biology-related information processing*; *Microelectromechanical systems*; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Applied computing** → Health informatics;

Additional Key Words and Phrases: microfluidics, planar placement, mVLSI

ACM Reference format:
Brian Crites, Karen Kong, and Philip Brisk. 2017. Diagonal Component Expansion for Flow-Layer Placement of Flow-Based Microfluidic Biochips. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 126 (September 2017), 18 pages.
https://doi.org/10.1145/3126529

1 INTRODUCTION

This paper studies the problem of automated physical design (placement and routing) for the flow layers of microfluidic very large scale integration (mVLSI) chips. At present, both layers are manually designed using software such as SolidWorks or AutoCAD. Manual design is tedious, error-prone, and unlikely to scale as integration densities increase. 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 [12] and where standard cells have a uniform height and are placed in rows; thus, established physical design techniques cannot easily be adapted for mVLSI technology.

This article was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2017 and appears as part of the ESWEEK-TECS special issue. This research was supported by the National Science Foundation, under award #1351115, #1536026, and #1540757.
Authors' addresses: B. Crites, K. Kong, and P. Brisk, 900 University Ave, Riverside, CA 92521; emails: {bcrit001, kkong006, philip}@cs.ucr.edu.

https://doi.org/10.1145/3126529

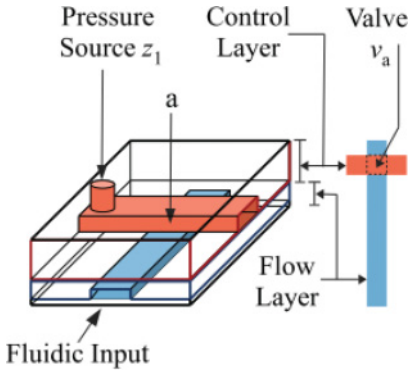


Fig. 1. Cross-section of a pressure-actuated microvalve fabricated using multilayer soft lithography [19].

Laboratories-on-a-chip (LoCs) based on continuous fluid flow microfluidics are widely used for a variety of biochemical applications. Through automation and miniaturization, LoCs offer the benefits of higher throughput, lower reagent usage, and reduced likelihood of human error compared to traditional benchtop chemistry methods. Although LoCs are widely used for biological research and are starting to gain traction in the diagnostics and biotechnology industries, they presently lack a design science comparable to the rapidly maturing field of electronic design automation (EDA).

Modern LoCs integrate hundreds or thousands of externally controllable *microvalves* [1, 5, 13, 19]. Figure 1 illustrates one representative microvalve technology based on *multi-layer soft lithography* [19]. Here, two layers of a flexible polymer substrate called *polydimethylsiloxane (PDMS)* are mounted on top of a rigid substrate (e.g., a glass slide): The *flow layer* on the bottom manipulates biological fluids of interest, while the *control layer* above provides actuation capabilities from an external pressure source. The microvalve is formed where a control channel on one layer crosses a flow channel on another layer. By default, all microvalves are open; pressurizing a control channel closes the microvalves that it drives. Larger components, such as pumps, mixers, switches, multiplexers and demultiplexers, memories, etc. can be constructed as an interconnected network of microvalves [13]. At present, successful multi-flow-layer mVLSI chips have not been demonstrated, requiring that the placement and routing of all flow components and connections be done on a single layer. Since inadvertent fluid mixing would invalidate any biological experiment, it is necessary for the components and channels to not overlap or intersect, a concept in graph theory known as graph planarity.

1.1 Motivating Example

Figure 2(a) shows an LoC that was designed manually and published in 2006 [20]. The approach advocated here is to start with a language-based specification of the chip, e.g., using *microfluidic hardware design language* (MHDL) [10], which is then compiled into a netlist, represented by a graph. To produce the layout, the first step is to compute a planar embedding of the graph as shown in Figure 2(b), 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(c), 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(d), while trying to maintain as much of the original planar embedding as possible. The final step is to

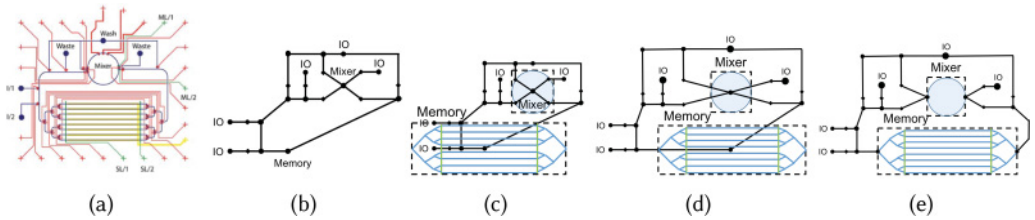


Fig. 2. (a) A programmable mVLSI LoC laid out manually [20]; (b) planar embedding of a mVLSI netlist representation of the LoC where no components or routes overlap or intersect, making it planar; (c) component expansion after planar embedding yields an illegal mVLSI layout, where several microvalves and IO ports overlap with the expanded memory and mixer components; (d) shifting the positions of expanded components yields a legal mVLSI layout, which is larger but similar to the original planar embedding; and (e) the legal mVLSI layout after routing and port recovery, with valid port-to-port connections.

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(e). This process produces a legal, although not necessarily optimal, flow layer design.

1.2 Contribution

This paper introduces a heuristic approach for component expansion which transforms a planar graph embedding (e.g., Figure 2(b)) into a legal mVLSI layout (e.g., Figure 2(d)) by judiciously shifting the location of vertices in the planar layout as each vertex is expanded into a component. This heuristic is evaluated in the context of an algorithmic design flow first proposed by McDaniel et al. [9]. The component expansion heuristic presented here improves mVLSI area utilization by a factor of $8.90x$ and reduces average fluid routing channel length by 47.4% compared to McDaniel et al.'s baseline.

2 PRELIMINARIES

2.1 mVLSI Netlist Representation

An *mVLSI netlist* $M = (C, E)$ is a set of components C and the fluid channels E that connect them. Typical mVSLI components include microvalves and input/output ports, as well as larger components constructed hierarchically from multiple microvalves and channels (e.g., mixers, memories, etc.). Each edge $(c_i, c_j) \in E$ represents a fluid channel that connects components c_i and c_j. This representation treats intersection points where multiple channels converge as components (switches), enabling a graph, rather than hypergraph-based, representation.

Component $c_i \in C$ is a tuple $c_i = (P_i, x_i, y_i, l_i, w_i)$: P_i is the set of ports (locations on the perimeter of c_i to which fluid channels can connect), (x_i, y_i) is the coordinate of the upper left corner of c_i after placement, and l_i and w_i are c_i's length and width; non-rectangular components, such as circular mixers, are approximated by rectangular bounding boxes. Each port $p_{i,j} \in P_i$ is located at position $(x_{i,j}, y_{i,j})$ on the perimeter of c_i. The dimensions of each component and locations of each port on its perimeter are provided in a component library.

Fluid channel $e_i \in E$ is a tuple $e_i = (c_j, p_{j,k}, c_m, p_{m,n}, S_i)$, which indicates that e_i connects adjacent components c_j and c_m through respective ports $p_{j,k} \in P_j$ and $p_{m,n} \in P_m$; S_i is a sequence of points representing a routing path from port $p_{j,k}$ to $p_{m,n}$.

During the layout process, *the placer* determines the location (x_i, y_i) of each component $c_i \in C$ while *the router* determines the path S_j for each edge $e_j \in E$; given placement information for each component, the location of the ports on its perimeter can be derived deterministically.

2.2 Planar Graphs and mVLSI Netlists

A graph can be viewed as a degenerate mVLSI netlist in which components are reduced to points, i.e., their areas and ports are ignored. A graph is *planar* if it can be *embedded* in the plane, i.e., if it can be drawn in such a manner that edges only cross at their endpoints. Embedding a planar graph provides the (x_i, y_i) coordinates for each component c_i and a routed path S_j for each edge e_j. Every planar graph admits a *straight line planar embedding* which enforces the property that each routed path S_j degenerates to a single line segment.

Planarity and straight-line planarity readily extends to mVLSI netlists. In a legal mVLSI embedding, placed components cannot overlap one another, routed fluid channels cannot intersect placed components or one another, and each routed fluid channel must connect to appropriate ports of its incident components.

2.3 Planar mVLSI Layout

This section summarizes a planar mVLSI embedding algorithm proposed by McDaniel et al. [9].

2.3.1 Planar Graph Embedding. The input is a mVLSI netlist $M = (C, E)$, which is initially treated as a graph in its degenerate form where components are points and ports are ignored. Non-planar graphs are first planarized by introducing switches and additional control lines [18]; all of our benchmarks are planar (as are real-world mVLSI LoCs), so our implementation presently omits this step. The next step is to compute a planar embedding of the graph; in principle, any algorithm can be used. McDaniel et al. use the Chrobak-Payne planar embedding algorithm [2], which has a publicly available implementation in the *Boost Library*[1]. This places each component c_i at location (x_i, y_i) in a two-dimensional grid.

2.3.2 Component Expansion. Components are sorted by their x-coordinates such that c_i is to the left of c_{i+1}, and are processed from left-to-right. Each component c_j, $j > i$ is shifted right by l_i to make room for c_i's horizontal expansion. The process then repeats in the vertical direction, with components shifted vertically by w_i. This process updates the (x_i, y_i) coordinates of each component and ensures that there is no overlap following placement.

2.3.3 Flow Channel Routing and Port Assignment. Simultaneous flow channel routing and port assignment is modeled as a min-cost network flow problem [22]. The problem formulation ensures that routed flow channels do not overlap or inadvertently intersect. Although we lack a formal proof that routability is guaranteed, this algorithm successfully routed all of the benchmarks used in our experiments. This process computes the routing path S_j for each edge e_j, including the choice of ports on each end of the path. While some components require specific ports to be assigned for functional reasons, many others can be connected using an arbitrary port, since they operate the same irrespective of the direction of the fluid flow. This routing method supports specific port pre-assignment by restricting the flow network to connect exclusively to a pre-assigned port; this feature was not necessary for the benchmarks used in this paper, i.e., in our experiments, all components allow the router to freely chose ports.

3 DIAGONAL COMPONENT EXPANSION (DICE)

This section introduces *DIagonal Component Expansion (DICE)*, which works far better in practice than the original component expansion algorithm outlined in Section 2.3.2. As it's name suggests, DICE tends to place components on a diagonal axis from the upper left corner of the chip to the

[1] http://www.boost.org.

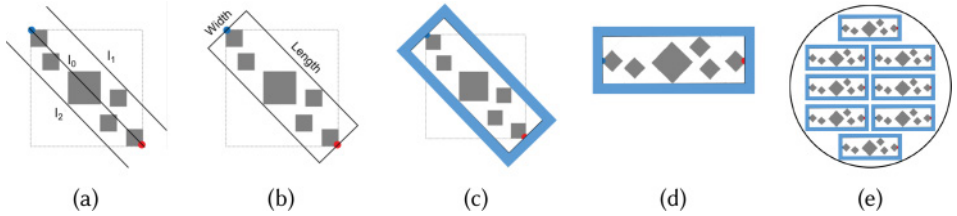


Fig. 3. (a) After diagonal expansion, line segment l_0 connects the upper-left corner of the leftmost component to the lower-right corner of the rightmost component; line segments l_1 and l_2, parallel to l_0, constrain the chip width; (b) the mVLSI chip is defined as a diagonal rectangle along the diagonal axis parallel to l_0; (c) the chip area expands (blue) after routing; (d) the chip is rotated so that it's length is parallel to the x-axis; (e) an mVLSI mold containing multiple copies of the chip layout.

lower right corner, yielding a compact, yet routable, layout. After a brief overview, we provide a detailed description of the DICE heuristic.

3.1 Overview

Figure 3 depicts the DICE process. Given a placed mVLSI chip with components expanded diagonally, let dl_0 be the diagonal line segment from the top left component's upper left corner to the bottom right component's bottom right corner, as shown in Figure 3(a). The length of dl_0 is the length of the placed (but not yet routed) mVLSI chip along the diagonal axis. The next step is to calculate upper and lower bound lines, dl_1 and dl_2, which are parallel to dl_0, and of equal length, as shown in Figure 3(b). This can be done quickly by iterating through the lower-left and upper-right corners of each component. Ignoring routing, the width of the resulting chip is the perpendicular distance between dl_1 and dl_2.

The last step is to route the fluid channels, trying to constrain them to the rectangular form factor along the diagonal axis. As shown in Figure 3(c), routing may increase the length and width; in the worst case, the expansion is proportional to $|E|$, the number of fluid channels and the spacing between them. The result remains rectangular. As shown in Figure 3(d), it is possible to rotate the chip layout to lie along the horizontal axis. This is allowable because mVLSI layout tools impose no constraints on the size, position, or rotational orientation of components. It is possible to "draw" the chip with either orientation in SolidWorks or AutoCAD.

Figure 3(e) depicts a mold containing multiple copies of the chip layout; this one can fabricate 8 copies of the same chip. Reducing the chip area increases the number of chips per mold; although a detailed description of mVLSI fabrication is beyond the scope of this paper, it suffices to note that the fabrication process itself lacks economies of scale.

3.2 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 4. 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.

3.3 Diagonal Expansion

Diagonal Expansion (pseudo code in Figure 1) tries to minimize the necessary increase in chip area due to component expansion.

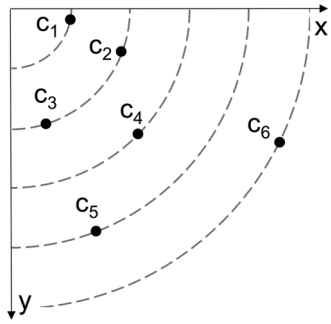


Fig. 4. Component selection by circular propagation. Components are selected in increasing order of subscript.

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 5(a)); 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 edge (Figure 5(b)). 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 5(c)). 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 5(d)). 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

ALGORITHM 1: Diagonal Expansion Algorithm

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

```
for c_i ∈ C do
    C ← C \ {c_j}
    Δ_x ← 0, Δ_y ← 0
    c_i.expand_component()
    for c_i ∈ C do
        if c_i.inside_left_or_above(c_j) then
            δ_x ← x_i + l_i − x_j
            δ_y ← y_i + w_i − y_j
            if δ_x > Δ_x then
                Δ_x ← δ_x
            end
            if δ_y > Δ_y then
                Δ_y ← δ_y
            end
        end
    end
    for c_i ∈ C do
        x_i ← x_i + Δ_x + Δ_buf
        y_i ← y_i + Δ_y + Δ_buf
    end
end
```

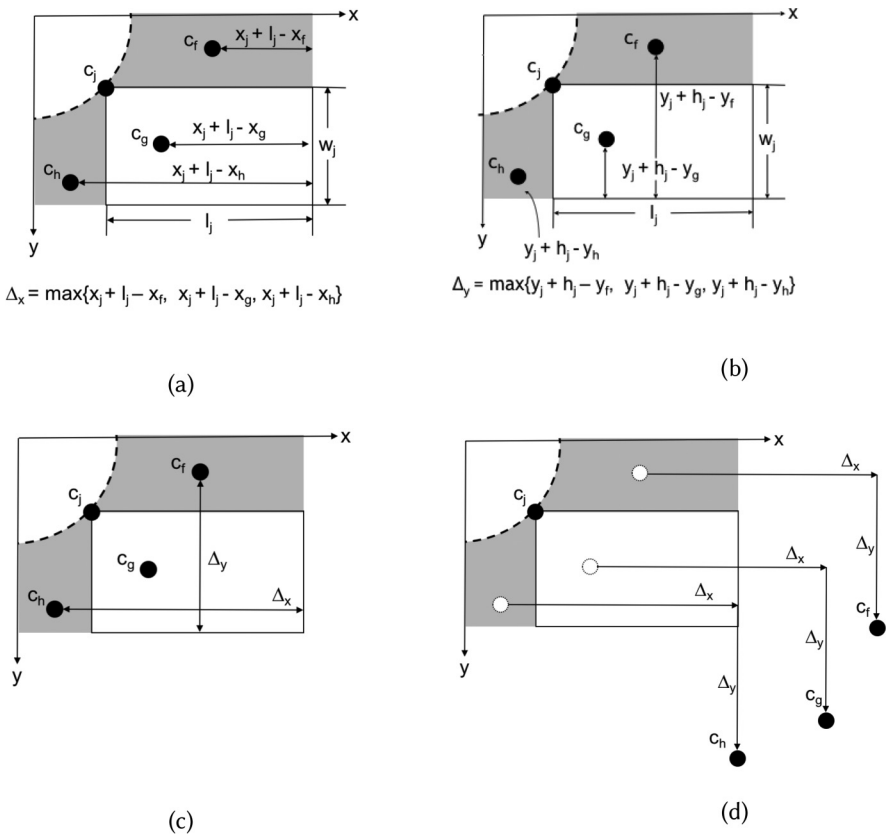


Fig. 5. DICE example: (a)-(c) Computation of Δ_x and Δ_y; (d) each non-expanded point is shifted accordingly.

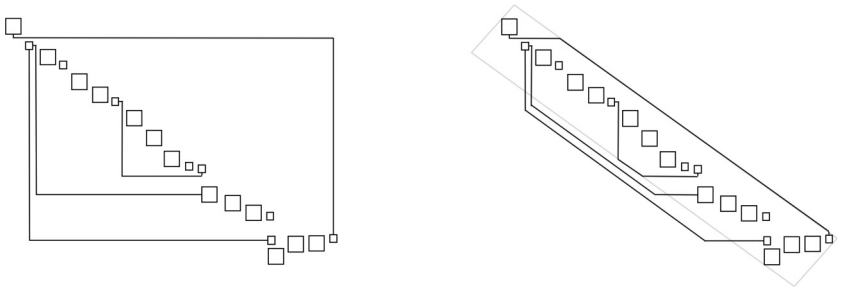
new coordinate for component c_i is $(x_i + \Delta_x + \Delta_{buf}, y_i + \Delta_y + \Delta_{buf})$. These shifts will spread the components along the device diagonal, leaving the majority of the component's ports unblocked by other components and free for use in routing.

3.4 Diagonally-Constrained Channel Routing

Figure 6(a) shows a legal fluid channel routing solution following DICE; only the routes that cross the boundary of the diagonal envelope from Figure 3(b) 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 6(b), the new routes run parallel to l_1 and l_2 while obeying spacing rules. If diagonal routes are not allowed, 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 [6]. From there, the resulting mVLSI chip can be cut and rotated as shown in Figure 3(c)



(a) (b)

Fig. 6. 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.

and 3(d). 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.

4 BASELINE HEURISTICS

This section introduces two alternative methods for component expansion which we use as baselines for comparison with DICE in the following section. When expanding a component, each of these methods attempts to identify a single expansion factor that can be applied to all not-yet-expanded points in the netlist. The methods produce inferior results, but are described here for completeness.

4.1 Shift Expansion

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 [9]. 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 7(a) and 7(b) 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 7(c)).

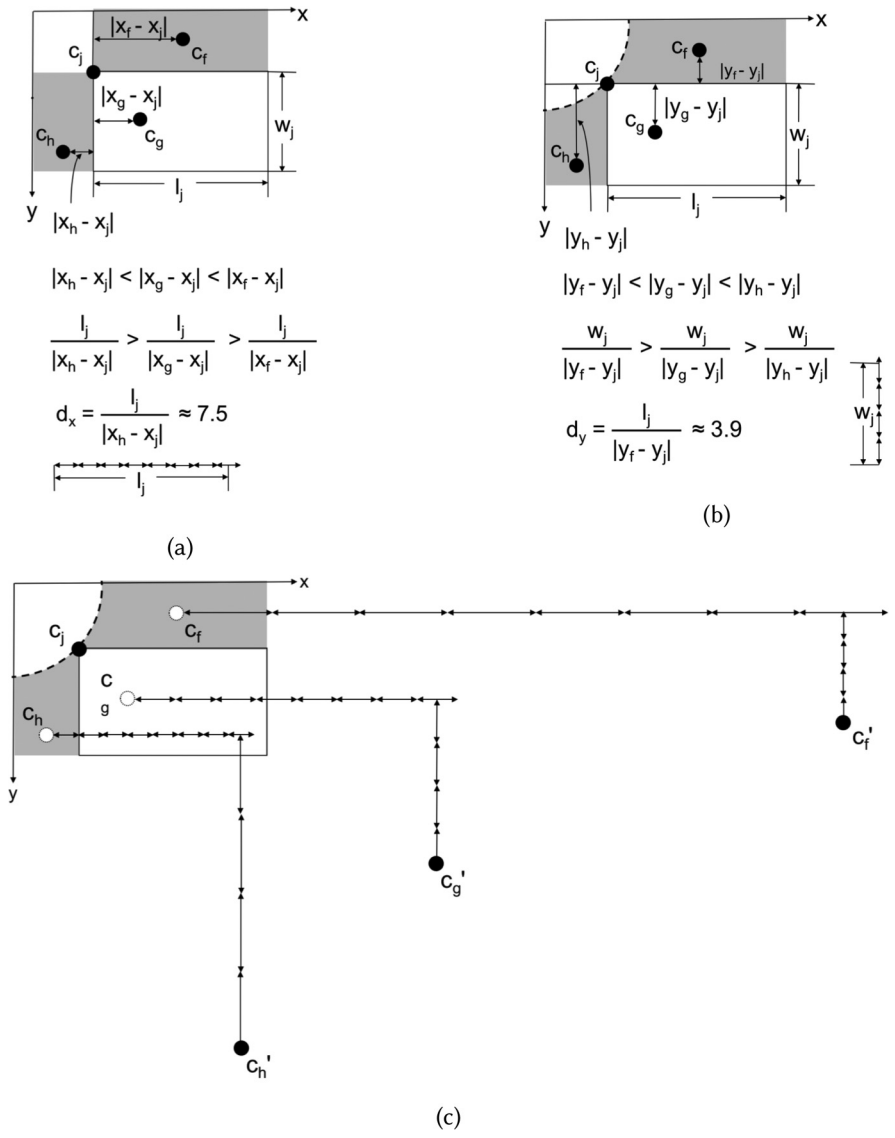


Fig. 7. Shifted Expansion example: (a)-(b) Computation of d_x and d_y; (c) each non-expanded point is shifted based on d_x, d_y and its distance from the expanding component.

4.2 Scaled Expansion

The *Scaled Expansion* expands upon Shifted Expansion 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.

5 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. In short, they are not nearly as important as prior papers that have emphasized optimization and (near-)optimality claim them to be.

5.1 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). One cannot understand the relationship between mVLSI area and cost without first understanding the fabrication process. The key takeways from the discussion which follows is that only dramatic reductions in mVLSI chip area will increase the number of chips per wafer, and, as we will discuss in the next section, mVLSI chip area has practically no impact on performance.

Each layer of an mVLSI chip is fabricated separately, and layers are only bonded together late in the fabrication process. The key objective of mVLSI fabrication is to imprint a pattern (flow and control channels) in each layer. This is done by creating a mold through a lithographic process, which sits atop a standard silicon wafer, typically 150mm or 200mm in diameter. Liquid polydimethylsiloxane (PDMS) is poured on top of the mold, after which it is spun (for consistency), degassed (to remove air bubbles), and baked (to solidify). The solidified PDMS is then removed from the mold, and the corresponding layer of each chip can be cut out, e.g., with an xacto knife (or an automatic cutter). For a multi-layer chip, the flow and control layers must then be aligned under a microscope and placed on a glass slide; then they are baked for bonding. Once the bonded chip is cooled, it is aligned, once again under a microscope, to punch I/O holes. Prior to use, the chip must be tested to ensure that it is defect-free.

This fabrication process lacks 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 publishable paper, 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

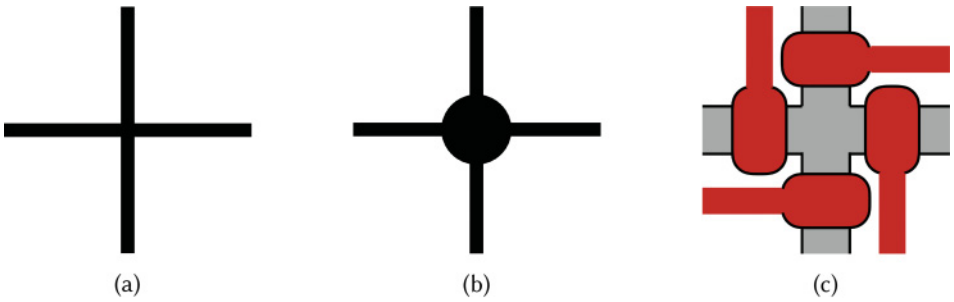


Fig. 8. Two intersecting edges resulting from a non-planar routing solution (a) can be planarized by inserting a vertex (b), which corresponds to a 4-valve mVLSI switch at the channel intersection point (c).

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 PDMS that is cut away 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 [18] or SAT solving [4].

5.2 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 is beyond the scope of this paper).

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.

5.3 Fluid Channel Intersections

Microfluidic chips are typically I/O limited, including both fluid and control. For example, the Stanford Microfluidic Foundry limits the number of I/Os to 35 as a design rule[2]. This has implications for layout constraints on mVLSI chips.

mVLSI chips are planar by definition. A non-planar layout can be planarized by inserting a switch at each point where two channels cross one another [18] (Figure 8); each switch requires four valves with two independent controls; a set of switches may share controls if they do not

[2]http://web.stanford.edu/group/foundry/Basic%20Design%20Rules.html.

operationally interfere with one another. The insertion of switches may negatively impact performance, as fluid flowing through switches must be serialized, and the switches must be decontaminated by rinsing after each fluid transfer. Thus, it is generally preferable to minimize the number of switches that must be inserted during layout, even within the scope of foundry-imposed design rules.

Our experimental evaluation considers planar netlists exclusively[3]. In our opinion, any layout method that inserts switches should be considered inferior to one that does not, before considering any other optimization criteria other than functional correctness.

6 EXPERIMENTAL RESULTS

DICE was implemented in C++ using the algorithmic framework summarized in Section 2.3. The framework employs a *unitless* grid, which decouples the layout and design rule checking processes from the manufacturing resolution of any one specific mVLSI technology [12]. 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.

6.1 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. [11], 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 component expansion method proposed by McDaniel et al. [9] (summarized in Section 2.3.2).
- **Shift Expansion (ShiftEx)** replaces BaseEx's component expansion step with Shift Expansion (Section 4.1).
- **Scaled Expansion (ScaleEx)** replaces BaseEx's component expansion step with Scaled Expansion (Section 4.2).
- **DICE-Unconstrained (DICE-U)** replaces BaseEx's component expansion step with DICE (Section 3.4), implemented with unconstrained fluid channel routing (Figure 6(a)).
- **DICE**, implemented with diagonally-constrained fluid channel (Figure 6(b)).

6.2 Benchmarks

We obtained netlists for four real-world planar mVLSI chips 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:** proprietary mVLSI LoC netlists provided by Microfluidic Innovations, LLC.
- **HIV1:** a multi-layer PDMS chip that performs a bead-based HIV1 p24 sandwich immunoassay [7].
- **MGG:** a molecular gradients generator that can generate five concentration levels of a two-sample mixture [15].

[3]We are unaware of any prior work on mVLSI synthesis or physical design that has evaluated non-planar netlists. In fact, [18], which introduced a planarization method, nevertheless only evaluated planar netlists.

Table 1. Benchmark Scale and Number of Intersections

Algorithm	# Components	# Connections	SA [11]	BaseEX [9]	ShiftEx	ScaleEx	DICE-U	DICE
AquaFlex-3b	14	13	25	0	0	0	0	0
AquaFlex-5a	17	16	36	0	0	0	0	0
HIV1 [11]	13	12	21	0	0	0	0	0
MGG [9]	30	38	167	0	0	0	0	0
Synthetic 1	21	21	52	0	0	0	0	0
Synthetic 2	12	11	13	0	0	0	0	0
Synthetic 3	34	33	142	0	0	0	0	0
Synthetic 4	34	33	136	0	0	0	0	0
Synthetic 5	45	45	207	0	0	0	0	0

The five synthetic benchmarks were generated by compiling a set of publicly available DAG specifications[4] through an established mVLSI architectural synthesis flow [10, 14].

Experiments were run using a buffer of 5 grid spaces for each component. Legal planar mVLSI embeddings were obtained for all component expansion algorithms.

6.3 Results and Analysis

For each component expansion heuristic and benchmark, we report the number of channel intersections (Table 1), the area utilization (Figure 9(a): the ratio of component area to total chip area expressed as a percentage) and the average routing channel length (Figure 9(c)). Computation time of the framework is dominated by the routing and port assignment phase (Section 2.3); varying the component expansion heuristic made a negligible impact.

6.3.1 Channel Intersections. As was stated in Section 5.3, 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.

Table 1 reports the number of fluid routing channel intersections for each of the layout heuristics that we evaluated. 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. Consequently, few, if any, of the netlists produced by SA could be fabricated at Stanford, 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.

6.3.2 Area Utilization. Figure 9(a) 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 DICE's result and the best result of the remaining heuristics was

[4] https://sites.google.com/site/mlsibiochips/home.

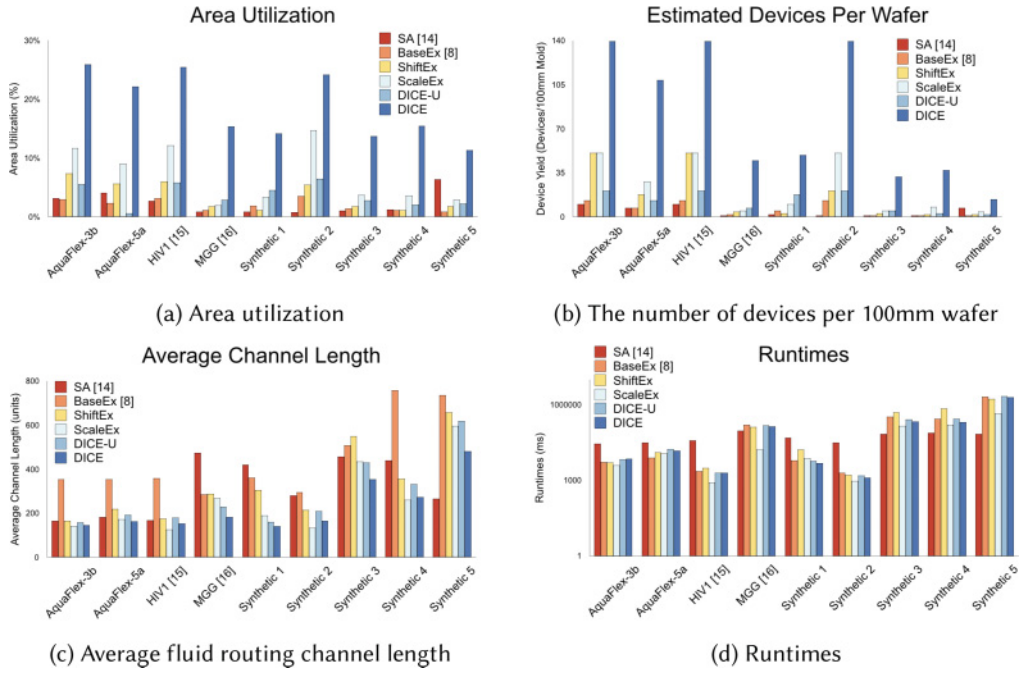


Fig. 9. Results reporting (a) total device are utilization, (b) the number of estimated devices that could be fabricated on a 100mm wafter, (c) the average length of a fluid routing channel, and (d) the runtimes of each algorithm on each benchmark.

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 peform 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 6 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, DICE's 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 diagonally-constrained routing. These observations explain why DICE achieves the highest area utilization reported in Figure 6.

6.3.3 Devices Per Wafer. Figure 9(b) reports the number of mVLSI flow layer devices that can be fabricated on a mold cast on a 100mm-diameter silicon wafer using the different layout algorithms. DICE achieves the highest number of devices per wafer for all benchmarks, with a significant gap between the second-best performer in each case. These results correlate more or less directly with the area utilization results shown in Figure 9(a).

6.3.4 Average Fluid Routing Channel Length. For all benchmarks in Figure 9(c), 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.

6.3.5 Runtime. Figure 9(d) 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. [11], 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 [22]. 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.

6.4 Case Study: AquaFlex-3b

As a case study, Figure 10 shows the flow layers of the AquaFlex-3b benchmark using all six placement and routing heuristics. SA (Figure 10(a)) yields a non-planar layout with por 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 10(b)) 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.

ShiftEx (Figure 10(c)) and ScaleEx (Figure 10(d)) 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 10(e)) 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 10(f)), which allows for diagonal routing parallel the diagonal axis, eliminates these inefficiencies, once the chip is cut out and rotated.

7 RELATED WORK

Prior work on mVLSI physical design falls into two categories: (1) layout of planar netlists, the focus of this paper; and (2) techniques to simultaneously planarize non-planar netlists while embedding them into the flow layer.

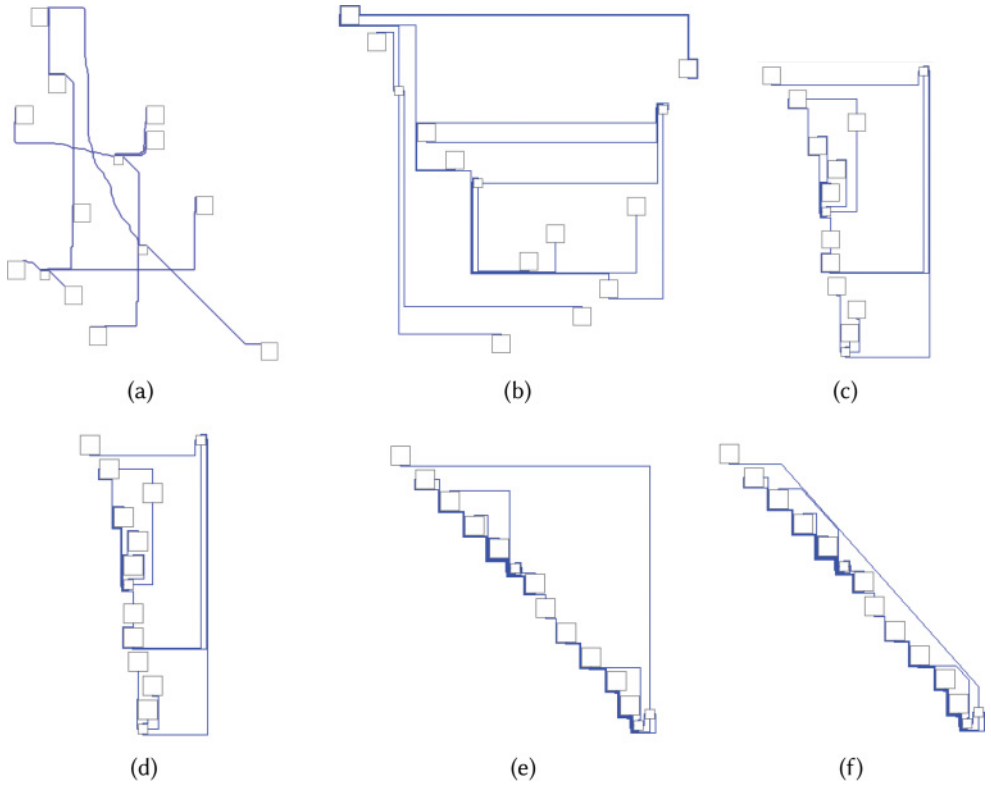


Fig. 10. A comparison of the layouts of AquaFlex-3b using (a) SA, (b) BaseEx, (c) ShiftEx, (d) ScaleEx, (e) DICE-U, and (f) DICE.

7.1 Planar mVLSI Netlist Embedding

The heuristics and experiments reported in this paper were conducted using an algorithm framework introduced by McDaniel et al. [9]. This framework consistently produces planar mVLSI embeddings without inserting additional switches, and our experiments confirm this observation. The compenent expansion heuristics introduced in this paper improve area utilization, average fluid channel length, and the number of devices per wafer.

Yao et al. [23] start with a planar graph embedding, improve it using simulated annealing, expand vertices into components, and use an A* router to route flow channels. Their framework admits non-planar flow channel crossings and does not perform port assignment either during or before routing.

Columba [18] models mVLSI physical design as an Integer Linear Program (ILP), which provides optimal solutions but may have scalability problems with large netlists. Columba co-optimizes the design of the flow and control layers of an mVLSI chip, which our algorithm does not. In the event that control layer layout fails after the flow layout is produced, we can re-generate the flow layers with relaxed placement constraints, as mentioned in Section 3.4, which will provide more flexibility to the control layer router. We believe that high runtimes associated with ILP solvers will limit scalability as netlist sizes increase. A more recent approach based on SAT solving, which generates only the flow layer, appears to suffer from similar scalability concerns [4].

7.2 Non-Planar mVLSI Netlist Embedding

Non-planar netlists can be planarized by placing multi-valve switches at intersection points between two crossing flow channels; however, this requires extra control lines, whose number is limited by modern design rules, such as those imposed by the Stanford Microfluidics Foundry. This limits the ability of physical design tools to synthesize non-planar netlists of arbitrary complexity. Non-planar mVLSI netlist synthesis may consider multiple objectives including minimizing the number of flow channel crossing, minimizing area, and minimizing flow channel length. Researchers have proposed placers using simulated annealing [11, 14, 21] and iterated cluster expansion (ICE) [17]. Non-planar routing algorithms may reduce fluid routing channel length, but do not limit the number of channel crossings [8]. These approaches are unlikely to yield workable mVLSI chips that can be fabricated, and should not be used to lay out planar mVLSI netlists.

The problem of simultaneous mVLSI planarization and layout with minimal switch insertion is identical to the problem of graph embedding while minimizing the number of edge crossings; the reduction inserts switches at each cross-point, as opposed to allowing a non-planar embedding. The decision problem of determining whether a graph can be embedded with at most K edge crossings is NP-complete [3]. The graph drawing community has studied this problem in great detail (e.g., see Ch. 2 of Ref. [16]); should non-planar mVLSI netlists become prevalent (e.g., due to widespread adoption of mVLSI architecture synthesis tools [10, 14], none of which guarantee planar netlists), then these algorithms represent a good starting point for future work on non-planar mVLSI layout.

8 CONCLUSION AND FUTURE WORK

This paper has investigated several techniques to convert a planar graph layout into a legal mVLSI netlist layout through the process of component expansion. Among the techniques considered, we have discovered that diagonal expansion works the best when coupled with diagonally constrained routing. That being said, two important issues remain open.

First, the experiments reported here use one specific planar embedding algorithm [2] provided by the Boost Library; further experimentation is needed to evaluate the quality of these component expansion techniques when used in conjunction with other planar graph embedding algorithms. Second, since planar graph embedding algorithms are based on the premise that vertices have no area, it may be possible to discover more effective direct placement techniques that do not start with an initial graph embedding. We plan to investigate both of these issues in the future.

REFERENCES

[1] Ismail Emre Araci and Stephen R. Quake. 2012. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. *Lab-on-a-Chip* 12, 16 (Aug. 2012), 2803–6. DOI: http://dx.doi.org/10.1039/c2lc40258k

[2] Marek Chrobak and Thomas H. Payne. 1995. A linear-time algorithm for drawing a planar graph on a grid. *Inform. Process. Lett.* 54 (1995), 241–246.

[3] M. R. Garey and D. S. Johnson. 1983. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods* 4, 3 (1983), 312–316. DOI: http://dx.doi.org/10.1137/0604033

[4] Andreas Grimmer, Qin Wang, Hailong Yao, Tsung-Yi Ho, and Robert Wille. 2017. Close-to-optimal placement and routing for continuous-flow microfluidic biochips. In *22nd Asia and South Pacific Design Automation Conference, ASP-DAC 2017, Chiba, Japan, January 16-19, 2017*. 530–535.

[5] William H Grover, Alison M. Skelley, Chung N. Liu, Eric T. Lagally, and Richard A. Mathies. 2003. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. *Sensors and Actuators B: Chemical* 89, 3 (2003), 315–323. DOI: http://dx.doi.org/10.1016/S0925-4005(02)00468-9

[6] Akihiro Hashimoto and James Stevens. 1971. Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the 8th Design Automation Workshop (series: DAC)*. 155–169. DOI: http://dx.doi.org/10.1145/800158.805069

[7] Baichen Li, Lin Li, Allan Guan, Quan Dong, Kangcheng Ruan, Ronggui Hu, and Zhenyu Li. 2014. A smartphone controlled handheld microfluidic liquid handling system. *Lab-on-a-Chip* 14 (2014), 4085–4092. Issue 20. DOI: http://dx.doi.org/10.1039/C4LC00227J

[8] Chun-Xun Lin, Chih-Hung Liu, I-Che Chen, D. T. Lee, and Tsung-Yi Ho. 2014. An Efficient Bi-criteria Flow Channel Routing Algorithm For Flow-based Microfluidic Biochips. In *Proceedings of the The 51st Annual Design Automation (DAC)*. 141:1–141:6. DOI: http://dx.doi.org/10.1145/2593069.2593084

[9] Jeffrey McDaniel, Brian Crites, Philip Brisk, and William H. Grover. 2015. Flow-Layer Physical Design for Microchips Based on Monolithic Membrane Valves. *IEEE Design & Test* 32, 6 (2015), 51–59. DOI: http://dx.doi.org/10.1109/MDAT.2015.2459699

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

[11] Jeffrey McDaniel, Brendon Parker, and Philip Brisk. 2014. Simulated annealing-based placement for microfluidic large scale integration (mLSI) chips. In *Proceedings of the 22nd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 213–218.

[12] Carver Mead and Lynn Conway. 1980. *Introduction to VLSI Systems*. Addison-Wesley.

[13] Jessica Melin and Stephen R. Quake. 2007. Microfluidic large-scale integration: The evolution of design rules for biological automation. *Annual Review of Biophysics and Biomolecular Structure* 36 (Jan. 2007), 213–31. DOI: http://dx.doi.org/10.1146/annurev.biophys.36.040306.132646

[14] Wajid Hassan Minhass, Paul Pop, Jan Madsen, and Felician Stefan Blaga. 2012. Architectural Synthesis of Flow-Based Microfluidic Large-Scale Integration Biochips. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES)*. 181–190.

[15] Minsoung Rhee and Mark A. Burns. 2008. Microfluidic assembly blocks. *Lab-on-a-Chip* 8 (2008), 1365–1373. Issue 8. DOI: http://dx.doi.org/10.1039/B805137B

[16] Roberto Tamassia (Ed.). 2013. *Handbook on Graph Drawing and Visualization*. Chapman and Hall/CRC. https://www.crcpress.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/9781584884125.

[17] Kai-Han Tseng, Sheng-Chi You, Jhe-Yu Liou, and Tsung-Yi Ho. 2013. A Top-Down Synthesis Methodology for Flow-Based Microfluidic Biochips Considering Valve-Switching Minimization. In *Proceedings of the International Symposium on Physical Design (ISPD)*. 123–129.

[18] Tsun-Ming Tseng, Mengchu Li, Bing Li, Tsung-Yi Ho, and Ulf Schlichtmann. 2016. Columba: Co-layout synthesis for continuous-flow microfluidic biochips. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 147.

[19] Marc Alexander Unger, Hou-Pu Chou, Todd Thorsen, Axel Scherer, and Stephen R Quake. 2000. Monolithic Microfabricated Valves and Pumps by Multilayer Soft Lithography. *Science* 288, 5463 (April 2000), 113–116. DOI: http://dx.doi.org/10.1126/science.288.5463.113

[20] John Paul Urbanski, William Thies, Christopher Rhodes, Saman Amarasinghe, and Todd Thorsen. 2006. Digital microfluidics using soft lithography. *Lab-on-a-Chip* 6 (2006), 96–104. Issue 1. DOI: http://dx.doi.org/10.1039/B510127A

[21] Qin Wang, Yizhong Ru, Hailong Yao, Tsung-Yi Ho, and Yici Cai. 2016. Sequence-pair-based placement and routing for flow-based microfluidic biochips. In *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 587–592. DOI: http://dx.doi.org/10.1109/ASPDAC.2016.7428075

[22] Martin D.F. Wong, Hua Xiang, and Xiaoping Tang. 2003. Min-cost flow-based algorithm for simultaneous pin assignment and routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 7 (July 2003), 870–878. DOI: http://dx.doi.org/10.1109/TCAD.2003.814258

[23] Hailong Yao, Qin Wang, Yizhong Ru, Yici Cai, and Tsung-Yi Ho. 2015. Integrated Flow-Control Codesign Methodology for Flow-Based Microfluidic Biochips. *IEEE Design & Test* 32, 6 (2015), 60–68. DOI: http://dx.doi.org/10.1109/MDAT.2015.2449180

Received March 2017; revised June 2017; accepted June 2017