URBER: Ultrafast Rule-Based Escape Routing Method for Large-Scale Sample Delivery Biochips

Jiayi Weng Tsung-Yi Ho Weiqing Ji Peng Liu Mengdi Bao Hailong Yao

***Abstract*—In high-throughput drug screening applications, as manual drug sample delivery is time-consuming and error-prone, there is an urgent need for accurate and efficient drug sample delivery biochip for large-scale microwell arrays. This paper proposes a new microfluidic biochip architecture, where drugs are automatically prepared with different concentration values, and then delivered into multiple microwells. For large-scale drug sample delivery biochips, the routing of drug sample delivery channels is a very challenging task without effective routing solutions. This paper proposes an ultrafast rule-based escape routing method, called URBER, to address the large-scale routing of drug sample delivery channels, which scales well in both runtime and memory even for a very large problem size. URBER runs very fast because it routes channels based on a set of pre-defined rules, which avoids runtime consumed in solution space exploration. All benchmarks for $30 \leq N, M \leq 100$ have been tested, where N and M are the number of columns and rows of the terminal array. Among these benchmarks, about ∼91.9% are routed with optimal solutions, and the runtime is order of magnitudes faster than optimal min-cost flow-based methods (speedup is from ∼600 to ∼340k). Specifically, for all benchmarks with $\frac{M}{N} \in (\frac{3}{4}, \frac{4}{3})$, optimal routing solutions are always obtained. URBER also shows promise of routing large-scale designs with up to 500k terminals efficiently.**

***Index Terms*—Microfluidic biochips, Drug delivery, Sample delivery, Rule-based routing, Microwell array, Escape routing**

I. Introduction

In various biochemical applications, different samples/drugs of different concentration values need to be delivered into different microwells, such as single-cell genome sequencing [1], cell culture [2], cell analysis [3], drug screening [4], etc. Current methods for delivering multiple drugs include manual pipetting apparatus and automated pipetting robot. On one hand, the process of manual pipetting is time-consuming, laborious, and error-prone. On the other hand, the automated pipetting robot is only suitable for volume production and is too expensive for researchers in laboratories. In contrast, automated drug sample delivery by microfluidic biochips enjoys notable advantages: (1) drug sample delivery biochips can be easily integrated into automatic point-of-care analyzers, (2) automatic sample preparation module can be integrated to deliver samples with different concentration values, and (3) different samples/reagents can be simultaneously delivered into different microwells without human intervention. Therefore, drug sample delivery microfluidic biochips are promising in enhancing the portability, accuracy, and automation of microwell-based biochemical applications.

In the past decade, different design automation methods have been presented for flow-based microfluidic biochips, including architectural synthesis and resource binding methods [5]–[9], flow-layer placement and routing methods [10]–[12], control optimization methods [13]–[20], co-design of both control and flow layers [21]–[25], storage optimization methods [26], [27], fault modeling and testing methods [28]–[31], random design methods [32], [33], etc.

However, none of the existing works on microfluidic biochips can be used to handle the large-scale drug sample delivery channel routing problem.

Although drug-delivery routing problem looks similar to the routing problem for printed circuit boards (PCBs), there are critical differences. Existing PCB routing methods address different routing problems with complex constraints, such as stagger-array design, differential-pair constraint, missing-pin design, and multilayer boards [34]–[36]. Although these methods can be modified to solve a drug sample delivery problem, either efficiency or optimality is not satisfactory. This is because existing methods are either based on integer linear programming (ILP) [37], which runs slow, or based on network flow formulation, which is sub-optimal due to the post-routing process [38]. We find out that due to the special characteristics of drug sample delivery problem, a much faster routing method is required.

In this paper, we propose an ultrafast rule-based escape routing method, called URBER, to address the scalability challenge in designing large-scale drug sample delivery biochip. Major contributions are as follows:

- The large-scale drug sample delivery biochip architecture is described, and the challenging large-scale escape routing problem is identified and effectively solved.
- The first ultrafast rule-based escape routing method is proposed, which conducts routing based on a set of prespecified rules, and thus significantly reduces runtime and enhances scalability to large-scale designs.

The work of H. Yao was supported in part by the National Natural Science Foundation of China (61674093). The work of T.-Y. Ho was supported in part by the Ministry of Science and Technology of Taiwan, under Grant MOST 105-2221-E-007-118-MY3 and 104-2220-E-007-021 and in part by the Technical University of Munich-Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Program under grant agreement no 291763.

Jiayi Weng, Weiqing Ji, Mengdi Bao, and Hailong Yao are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.

Tsung-Yi Ho is with the Department of Computer Science, National Tsing Hua University.

Peng Liu is with the Department of Biomedical Engineering, School of Medicine, Collaborative Innovation Center for Diagnosis and Treatment of Infectious Diseases, Tsinghua University, Beijing 100084, China.

Corresponding author: Hailong Yao, E-mail: hailongyao@tsinghua.edu.cn.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2883908, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

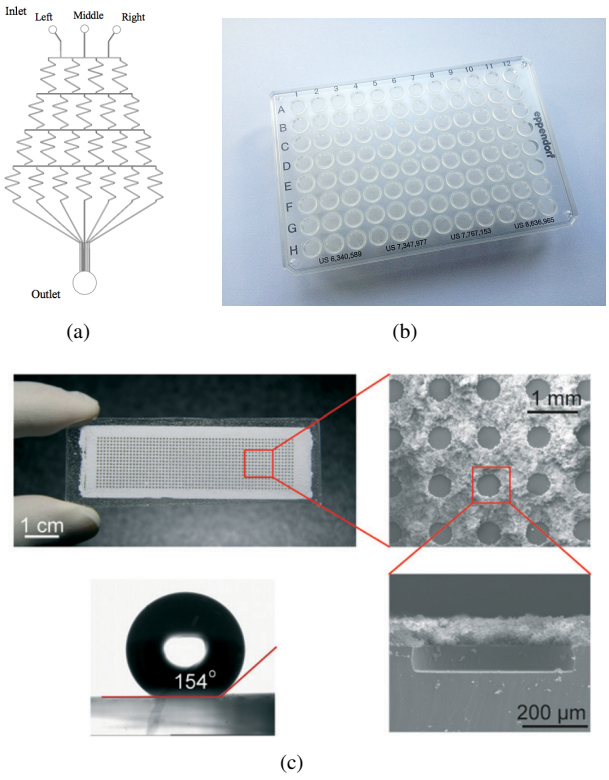


Fig. 1. (a) Concentration gradient generator [39], (b) a 96-well plate, and (c) close look at a microwell array [3].

- URBER is not only orders of magnitude faster than optimal min-cost flow-based routing method, but also obtains optimal routing solutions in ∼91.9% cases among more than 5,000 benchmarks obtained by sweeping the benchmark sizes.

The remainder of this paper is organized as follows. Section II describes the proposed biochip architecture along with problem formulation. Section III presents the rule-based regular routing method. Section IV presents and discusses experimental results. Finally, a conclusion is drawn in Section V.

II. Biochip Architecture and Problem Formulation

A. Biochip Architecture

Figure 1 shows examples of concentration gradient generator (Figure 1(a)) and w-well plate (Figure 1(b)). In proposed drug sample delivery biochip architecture, w different types of drugs are loaded from the w-well plate, and then different concentration values for each input drug are obtained by concentration gradient generator. For each concentration value, multiple copies of fluids are obtained and then delivered to the microwells (Figure 1(c)). For the concentration gradient generator shown in Figure 1(a), individual drugs are introduced into the middle inlet and water into two outer inlets. The gradient concentrations are generated based on repeated mixing, splitting, and recombination of laminar-flow fluids using a serpentine network of microfluidic channels [40].

Figure 2 shows a schematic diagram of proposed microfluidic biochip design. w drugs preserved in a w-well plate (Figure 1(b)) are dispensed into w concentration gradient generators for diluting each drug into v concentrations, and then each concentration is copied k times. Finally, different copies are delivered into microwell array for drug screening applications. This biochip architecture requires only a single polydimethylsiloxane (PDMS) layer to finish complex sample dilution and delivery tasks.

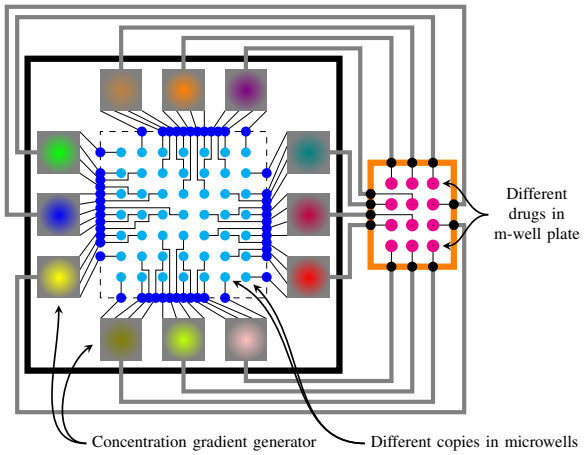


Fig. 2. Large-scale drug sample delivery microfluidic biochip design: w drugs in w-well plate are dispensed into w concentration gradient generators for diluting each drug into v concentrations, and then each concentration is copied k times for drug screening.

B. Problem Formulation

As shown in Figure 2, a large number of different drug copies are required to be routed to specified microwell positions. For example, for a target microfluidic biochip design with 384-well plate, assuming the number of concentration values for each drug is $v = 5$, and the number of copies for each drug sample is $k = 10$. Then there are in total $384 \times 5 \times 10 = 19200$ microwells to be routed simultaneously. Existing escape routing methods never addressed such large routing problems [38], [41], [42]. Moreover, to enable high-throughput drug screening, even larger designs are required, which bring great challenges to automatic design methods. In this paper, we propose the ultrafast rule-based escape routing method, which can obtain high-quality design solution for above design problem within few seconds.

The multiple drug sample delivery problem can be stated as follows:

Given: w drugs from the w-well plate, w gradient concentration generators, and high-throughput microwell array.

Find: The escape routing solution for drug sample delivery microfluidic biochip, which simultaneously delivers diluted drug samples from external terminals (outputs of gradient concentration generators) to internal terminals (microwells).

Constraints: (1) Each internal terminal of the biochip is aligned to a microwell; (2) Each external terminal is connected to an output port of the gradient concentration generator; (3) Design rules such as minimum spacing and channel width are satisfied.

Objective: Minimize total channel length.

III. Ultrafast Rule-Based Escape Routing Method

Definition 1 (Routing Region): The routing region is defined as the rectangular routing area enclosing internal nodes, which are connected to the concentration gradient generator.

TABLE I
NOTATIONS USED IN THE PROPOSED METHOD.

S	Internal terminal
T	External terminal
N/M	Number of internal terminals in horizontal/vertical direction
$S[n,m]$	The internal terminal at n^{th} column and m^{th} row
d	Number of routing pitches between adjacent internal terminals
d_0	Minimum feasible value of d for successful routing
g_x/g_y	Total horizontal/vertical routing grids in routing region
R_i	Routing subregion i
$\|R_i\|$	Number of internal terminals in subregion i
B	Boundary line of the routing region
$D_M/-D_M$	Main direction/the opposite direction of D_M
$D_A/-D_A$	Auxiliary direction/the opposite direction of D_A
n_x/n_y	Maximum column/row number of unrouted internal terminals in R_0
$T_x(t_x,0)$/ $T_y(0,t_y)$	Candidate external terminals in lower/upper triangular region as defined in Definition 13
S_x/S_y	Candidate internal terminal in lower/upper triangular region
c_x/c_y	Candidate routing channel in lower/upper triangular region

Definition 2 (Internal Terminal): An internal terminal is defined as the internal node corresponding to microwell for drug sample delivery.

Definition 3 (External Terminal): An external terminal is defined as the terminal at the boundary of the routing region interconnecting internal terminal and concentration gradient generator.

Definition 4 (Routing Pitch): The routing pitch is defined as the distance between centerlines of adjacent routing channels, which observes the minimum channel width and spacing constraints.

Definition 5 (Routing Grid): A routing grid is the intersection point between horizontal and vertical lines partitioned according to given routing pitch.

Definition 6 (Routing Channel): A routing channel is defined as a sequence of routing grids from an internal terminal to an external terminal.

Definition 7 (Axis): An axis (x-axis/y-axis) is built upon given routing region.

Figure 3 gives a routing example from the rule-based routing method within a rectangular routing region. In this figure, the internal terminal, external terminal, routing grid, and routed channel along routing grids are marked. The external terminals are on the boundary lines of the routing region. Axes are built as shown in the figure. Number of horizontal internal terminals $N = 6$, number of vertical internal terminals $M = 4$, number of routing pitches between internal terminals $d = 2$, the minimum feasible value of d for successful routing $d_0 = 2$, and the total number of routing grids $(g_x \times g_y) = (14 \times 10)$. Table I shows the notations used in our proposed method.

Definition 8 (Central Axis): The central h-axis and central v-axis are defined along center lines of entire routing region. For total g_x/g_y routing grids along horizontal/vertical direction, the central v-axis/h-axis is at $x = g_x/2$ / $y = g_y/2$.

Definition 9 (Routing Subregion): The routing subregion is defined as the partial routing area of the original routing region after partitioning by the divide-and-conquer approach.

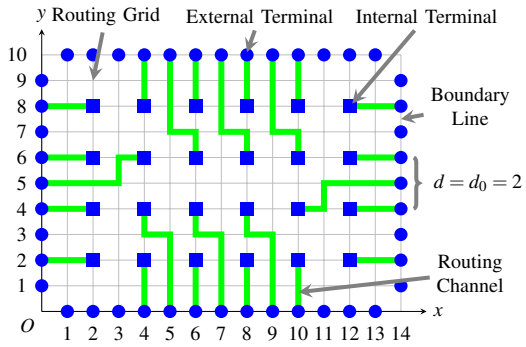


Fig. 3. A routing example.

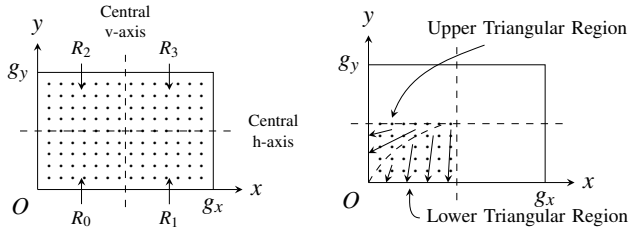


Fig. 4. Partitioning of routing region: (a) routing subregions with central h-axis and central v-axis, and (b) upper and lower triangular regions within one routing subregion R_0.

For each internal terminal S_i, it can be connected to external terminal T_j only if S_i and T_j belong to the same subregion R_k.

Definition 10 (Upper/Lower Triangular Region): During routing in a subregion, each internal terminal is assigned to either lower or upper triangular region by the routing method, resulting in partitioning of two triangular regions. An internal terminal assigned to the upper/lower triangular region must be routed to an external terminal along corresponding y-axis/x-axis boundary line.

Figure 4 gives an example of partitioning entire routing region. As shown in Figure 4(a), the routing region is partitioned into 4 routing subregions, i.e., R_0, R_1, R_2, and R_3. In following sections, subregion R_0 consisting of routing grid $(0,0)$ is used for describing the proposed routing method. Here, g_x and g_y give the total number of routing grids along x and y direction, respectively. As shown in Figure 4(b), one routing subregion can be further partitioned into upper and lower triangular regions by applying proposed routing rules. As denoted by arrowed lines, internal terminals of different triangular regions can only be routed to their corresponding external terminals along y-axis/x-axis boundary lines.

A. Routing Flow

Figure 5 shows the overall flow of URBER. Given the number of internal terminals (N,M) and the number of routing pitches between adjacent internal terminals d, we first calculate total length g_x and total width g_y computed by Equation (1) (see Section III-B), corresponding to total number of routing grids in horizontal and vertical directions, respectively. Then the entire routing region is partitioned into 4 subregions by proposed *terminal assignment rule* as described in Section III-C. Next, the proposed routing rule *Rule-Central* is applied to route central-state internal terminals in all subregions as described in Section III-F1. When central-state terminals are

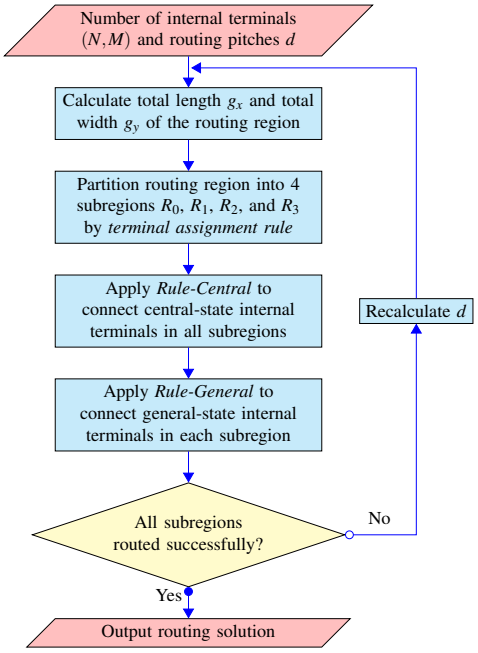


Fig. 5. Proposed routing flow of URBER.

routed, the proposed routing rule *Rule-General* in Section III-F2 is applied to route remaining internal terminals in each subregion. If there are routing failures, this routing process will be iterated with an increased value of d. Section III-I describes how to efficiently determine the feasible value of d. When all internal terminals are successfully routed, the routing channels will be output as the final solution. Section III-J presents the time complexity analysis.

B. Computation of Routing Grids

First, the number of internal terminals in horizontal and vertical directions, N and M, are loaded into the routing system, respectively. Given the number of routing pitches between adjacent internal terminals d, the total number of routing grids in horizontal/vertical direction g_x/g_y can be computed as

$$\begin{aligned} g_x &= (N+1)\times d \\ g_y &= (M+1)\times d \end{aligned} \tag{1}$$

For an internal terminal at the n^{th} column and m^{th} row, denoted as $S[n,m]$, its corresponding mapped coordinates (x,y) in partitioned routing grids are computed as

$$\begin{aligned} x &= n\times d \\ y &= m\times d \end{aligned} \tag{2}$$

C. Partitioning of the Routing Region

We propose a divide-and-conquer approach to speed up the routing process, which partitions complete routing region into different subregions. As subregions are similar to each other with respect to central v-axis and central h-axis, the symmetric property is utilized for routing speedup. So not all subregions need to be directly routed, and some of them can be resolved by coordinate transformation.

As shown in Figure 4(a), the entire rectangular routing region is divided into 4 rectangular subregions along central v-axis and central h-axis, respectively. Since routing subregions are similar to each other, we initially route internal terminals in R_0. And then by flipping routing solution around central h-axis and central v-axis, similar routing solutions in other routing subregions may be obtained, which avoids redundant computation of similar routing channels.

Given the number of internal terminals in horizontal and vertical directions, N and M, respectively, the routing subregion of i^{th} internal terminal $S_i[n_i,m_i]$ at $[n_i,m_i]$ can be obtained by following *terminal assignment rule*:

1) $n_i < (N+1)/2$, $m_i < (M+1)/2$: $S_i \in R_0$.
2) $n_i > (N+1)/2$, $m_i < (M+1)/2$: $S_i \in R_1$.
3) $n_i < (N+1)/2$, $m_i > (M+1)/2$: $S_i \in R_2$.
4) $n_i > (N+1)/2$, $m_i > (M+1)/2$: $S_i \in R_3$.
5) $n_i = (N+1)/2$, $m_i < (M+1)/2$: if $m_i \bmod 2 = 1$, $S_i \in R_0$; otherwise $S_i \in R_1$.
6) $n_i = (N+1)/2$, $m_i > (M+1)/2$: if $(M-m_i) \bmod 2 = 1$, $S_i \in R_2$; otherwise $S_i \in R_3$.
7) $n_i < (N+1)/2$, $m_i = (M+1)/2$: if $n_i \bmod 2 = 1$, $S_i \in R_2$; otherwise $S_i \in R_0$.
8) $n_i > (N+1)/2$, $m_i = (M+1)/2$: if $(N-n_i) \bmod 2 = 1$, $S_i \in R_3$; otherwise $S_i \in R_1$.
9) $n_i = (N+1)/2$, $m_i = (M+1)/2$: if $(N \geq M \wedge M \bmod 4 = 1) \vee (N < M \wedge N \bmod 4 = 3)$, $S_i \in R_0$; otherwise $S_i \in R_1$.

Figure 6 shows an example of applying above terminal assignment rule to internal terminals for $N = 7$ and $M = 5$. After applying terminal assignment rule, four routing subregions are formed by partitioned internal terminals. There are in total $7 \times 5 = 35$ internal terminals. Specifically, the number of internal terminals in each subregion is as follows: $|R_0| = 9$, $|R_1| = 9$, $|R_2| = 9$, $|R_3| = 8$. As shown in the figure, R_1 and R_2 are symmetrical. So we only need to route one subregion (either R_1 or R_2), and then flip routing solution to another subregion by coordinate transformation. For different values of N and M, different pairs of subregions are symmetrical. From the figure, the shape of partitioned subregions are not exactly a rectangle, but a rectangular shape with zigzag boundary edges.

Lemma 1: When applying terminal assignment rule, all internal terminals are assigned to a single subregion. Let $|R_i|$ denote the number of internal terminals in subregion R_i. Then, $\forall i,j \in \{0,1,2,3\}, \big||R_i| - |R_j|\big| \leq 1$.

Lemma 2: When applying terminal assignment rule, the following conditions hold: (1) when N and M are both odd numbers, there is only one pair of subregions with identical number of internal terminals, either $|R_0| = |R_3|$ or $|R_1| = |R_2|$; (2) when N or M is even number, there are two pairs of subregions with identical number of internal terminals ($|R_0| = |R_3|$ and $|R_1| = |R_2|$).

Lemma 3: When applying terminal assignment rule, if $|R_i| = |R_{3-i}|$, subregions R_i and R_{3-i} are symmetrical to each other, and the optimal routing solution of one subregion can be transformed to another subregion without loss in optimality.

D. Different Routing States

During the routing process, different internal terminals are not only assigned to different subregions, but also assigned to

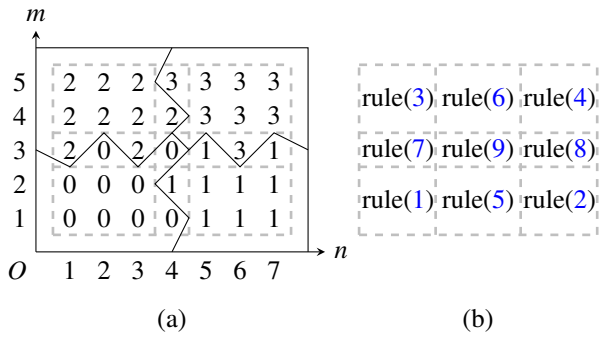


Fig. 6. Partitioning of routing region with $N = 7$ and $M = 5$: (a) total 4 subregions obtained from 9 blocks, where the internal terminals marked by number i ($i \in \{0,1,2,3\}$) belongs to subregion R_i, and (b) the correspondance between the 9 blocks and the 9 terminal assignment rules.

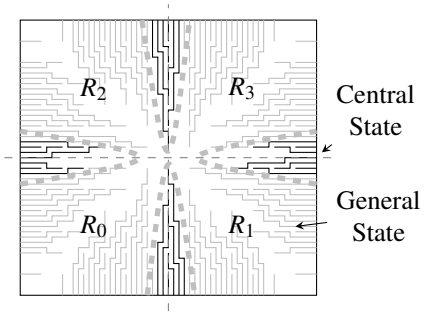


Fig. 7. Different routing states and routing subregions.

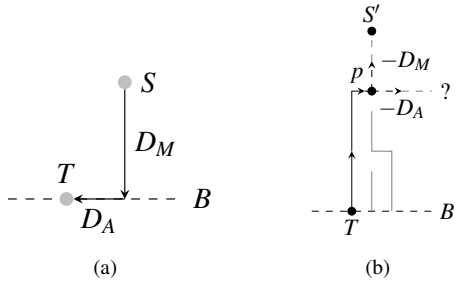


Fig. 8. Example of rule-based routing following main and auxiliary directions: (a) main and auxiliary directions, (b) when S' and p are collinear, p will be expanded along $-D_M$.

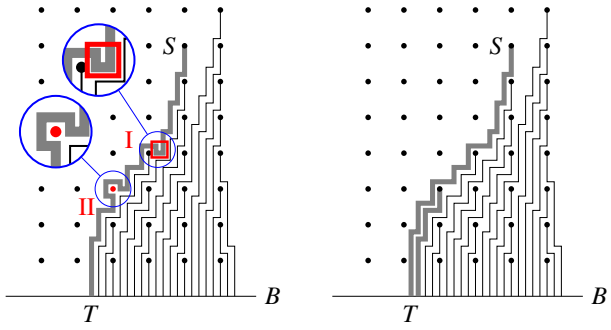
different *routing states* according to their specific positions. There are 2 routing states: *central state* and *general state*.

1) Central State: In the entire routing region, four set of internal terminals near central axes are assigned to the central state, whose corresponding external terminals are near the centers of corresponding boundary lines. Whether an internal terminal belongs to central state is determined by the routing rule in Section III-F1.

2) General State: Apart from central-state internal terminals, the remaining four set of internal terminals are assigned to the general state. Figure 7 illustrates the assignment of internal terminals to different routing states. In Figure 7, routing channels for central-state internal terminals are shown in black lines, and routing channels for general-state internal terminals are in gray lines.

E. Basic Routing Rules

Given internal terminal S, external terminal T, and boundary line B where T is located, we give the following definitions for describing the proposed routing rules.



(a) Routing from S to T (b) Routing from T to S

Fig. 9. Merits of routing from T to S. (a) Drawbacks of routing from S to T: (I) Unnecessary detours occur with increased length; (II) Certain unrouted internal terminals will be blocked. (b) Drawbacks in (a) are avoided by routing from T to S.

Definition 11 (Main Direction): The main direction D_M of S is defined as the searching direction from S towards B. The opposite direction of D_M is denoted as $-D_M$.

Definition 12 (Auxiliary Direction): The auxiliary direction D_A of S is defined as the searching direction from S towards T, which is perpendicular to D_M. The opposite direction of D_A is denoted as $-D_A$. When S and T are collinear, $D_A = D_M$.

Figure 8(a) shows the main direction and auxiliary direction from S to T, where T is to the bottom left of S. In this figure, D_M is going downward, and D_A is going left.

1) Rule-1: The first routing rule, *Rule-1*, starts from external terminal T, and synthesizes a routing channel toward internal terminal S'. Significant improvement of routing completion rate can be obtained by routing from external terminals rather than from internal terminals. Figure 9 gives an example illustrating the difference of this two cases. When routing from T toward S', for each routing grid, the direction of $-D_A$ is first checked. If there is a routing obstacle, e.g., an internal terminal or a pre-routed path, the direction of D_M is checked for routing. When an unrouted internal terminal S is reached, a routed channel is obtained and the routing process is terminated. Here, S may be different from S'. In this way, it is guaranteed that a staircase-shaped channel can always be obtained from T to certain unrouted internal terminal S.

Algorithm 1 presents the basic routing rule Rule-1. In this algorithm, c denotes routing channel, p denotes current routing grid, q denotes next routing grid in direction $-D_M$, r denotes next routing grid in direction $-D_A$, and $p-D_M/p-D_A$ denotes next routing grid in direction $-D_M/-D_A$. In Line 3-12, the while-loop determines routing grid of candidate internal terminal by p, which is initially assigned as T. The routing grid p is first expanded along $-D_A$. During expansion, when an obstacle is reached along $-D_A$, the expanding direction will be switched to $-D_M$. When S' and p are collinear, p will be expanded along $-D_M$ (see Figure 8(b)). During the expanding process, if there are routing obstacles along both $-D_A$ and $-D_M$, then the routing process fails ($c \leftarrow \varnothing$). Otherwise, p will reach an unrouted internal terminal, and this internal terminal will be marked as candidate internal terminal S. Routing channel c is also obtained during this expanding process. In URBER, all channels are routed by Rule-1, which is mostly called in other routing procedures. As Rule-1 carries

Input: Internal terminal S' and candidate external terminal T
Output: Candidate internal terminal S and routed channel c from S to T

1 Obtain D_M and D_A from S' to T;
2 $p \leftarrow T$, $c \leftarrow \{T\}$;
3 **while** p *is not an unrouted internal terminal* **do**
4 $q \leftarrow p - D_M$, $r \leftarrow p - D_A$;
5 **if** r *is not occupied and Noncollinear*(p, S') **then**
6 $p \leftarrow r$;
7 **else if** q *is not occupied* **then**
8 $p \leftarrow q$;
9 **else**
10 $S \leftarrow p$, $c \leftarrow \varnothing$;
11 **return** S, c
12 $c \leftarrow c \cup \{p\}$;
13 $S \leftarrow p$;
14 **return** S, c

Algorithm 1: Rule-1

out routing along only two alternating directions from T to S, we have following lemma.

Lemma 4: For each routed channel c_i from internal terminal S_j to external terminal T_k, Rule-1 guarantees the length of c_i to be equal to the Manhattan distance between S_j and T_k.

2) Rule-2: The second routing rule, *Rule-2*, obtains routing channels for a given number of internal terminals within one column or row. These terminals are connected to nearest boundary line B by Rule-1. According to column/row index of the current internal terminal to be routed, different auxiliary directions are adopted during routing. As index value changes, the auxiliary direction switches back and force correspondingly for enhanced routability.

Algorithm 2 presents details of routing rule Rule-2, where δ denotes whether or not to switch routing direction (see Figure 10(a)), and t_l/t_r denotes the coordinate of candidate external terminal nearby/away from origin (0,0) (see Figure 10(b)). In Line 1, initial values of t_l, t_r are computed. In Line 3-19, the while-loop computes routing channels by Rule-1 from S to T_l or T_r, depending on the value of $(i+\delta) \bmod 2$. The routing process is iterated until the number of routed internal terminals is greater than lim or certain external terminal T (T_l or T_r) is already occupied. Finally, the routing channels are returned.

Figure 10(b) shows a routing example based on Rule-2. In the figure, channels in gray lines are intermediately computed in the while-loop for the case of $i < 5$ (Line 3-19 in Algorithm 2). For $i = 5$, if S and T are determined, because there are no other unrouted internal terminals nearer to T along D_M and D_A, the computed routing channel (denoted in arrowed lines) will be connected to S.

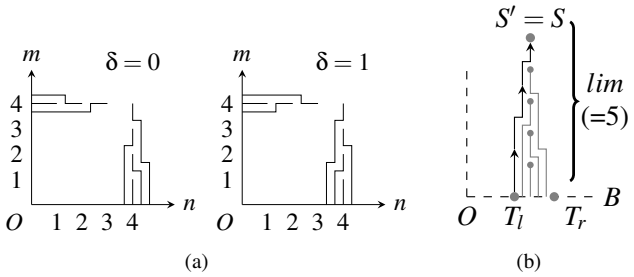
F. Rule-Based Routing for Different States

1) Rule-Central: The routing rule for central state, *Rule-Central*, connects central-state internal terminals using Rule-2 (see Section III-D1). Here, we only route two sets of internal terminals: (1) those along the boundary between R_0 and R_1, and (2) those along the boundary between R_0 and R_2. Then

Input: Column/Row $flag$, unrouted internal terminal's column/row number n, upper bound on the number of terminals to be routed lim, and switching flag δ
Output: Routed channels for given internal terminals

1 $t_l \leftarrow n \times d - \delta$, $t_r \leftarrow t_l + 1$;
2 $i \leftarrow 1$;
3 **while** $i \le lim$ *and* i^{th} *internal terminal is unrouted* **do**
4 **if** $flag$=*Column* **then**
5 $T_l \leftarrow (t_l, 0), T_r \leftarrow (t_r, 0), S \leftarrow (n \times d, i \times d)$;
6 **else**
7 $T_l \leftarrow (0, t_l), T_r \leftarrow (0, t_r), S \leftarrow (i \times d, n \times d)$;
8 **if** $(i+\delta) \bmod 2 = 1$ **then**
9 **if** T_l *is routed* **then**
10 **break**;
11 **else**
12 $c \leftarrow$ Rule-1(S,T_l), $t_l \leftarrow t_l - 1$;
13 **else**
14 **if** T_r *is routed* **then**
15 **break**;
16 **else**
17 $c \leftarrow$ Rule-1(S,T_r), $t_r \leftarrow t_r + 1$;
18 Store c as routing solution;
19 $i \leftarrow i + 1$;
20 **return** *Routed channels*

Algorithm 2: Rule-2



(a) (b)

Fig. 10. Routing example by Rule-2: (a) different routing solutions corresponding to different switching flags $\delta = 0$ and $\delta = 1$, and (b) routing solution from Rule-2.

coordinate transformation is performed to obtain the remaining routing solution as follows: (1) for internal terminals between R_1 and R_3, routing solution is obtained by transforming the solution of internal terminals between R_0 and R_2, and (2) for internal terminals between R_2 and R_3, routing solution is obtained by transforming the solution of internal terminals between R_0 and R_1. By coordinate transformation, the routing solution in each subregion maintains symmetric. Algorithm 3 presents details of routing procedure, and Figure 11 presents a routing example of applying Rule-Central.

In Algorithm 3, routing solution of central-state internal terminals is obtained by analyzing the odevity of N and M (see Line 3-7 and Line 8-12). If M is odd, there is only one row closest to central h-axis. The maximum number of internal terminals along this row, which can be successfully routed, is set to $2d-1$. As shown in Figure 11, as $M = 9$ (odd) and $d = 4$, there are maximum 7 $(2 \times 4 - 1)$ internal terminals that can be successfully routed along row 5 $((M+1)/2$ as given

Input: N, M, d, g_x and g_y.
Output: Routed channels of central-state internal terminals.

1 $l_n \leftarrow \lfloor N/2 \rfloor$;
2 $l_m \leftarrow \lfloor M/2 \rfloor$;
3 **if** $N \bmod 2 = 1$ **then**
4 | Rule-2(Column, $(N+1)/2$, $\min\{2d-1, l_m\}$, δ=0);
5 **else**
6 | Rule-2(Column, $N/2$, $\min\{d, l_m\}$, δ=0);
7 | Rule-2(Column, $N/2+1$, l_m, δ=1);
8 **if** $M \bmod 2 = 1$ **then**
9 | Rule-2(Row, $(M+1)/2$, $\min\{2d-1, l_n\}$, δ=1);
10 **else**
11 | Rule-2(Row, $M/2+1$, $\min\{d, l_n\}$, δ=1);
12 | Rule-2(Row, $M/2$, l_n, δ=0);
13 Perform centrosymmetric coordinate transformation by Equation (3);
14 **return** *Routed channels*

Algorithm 3: Rule-Central

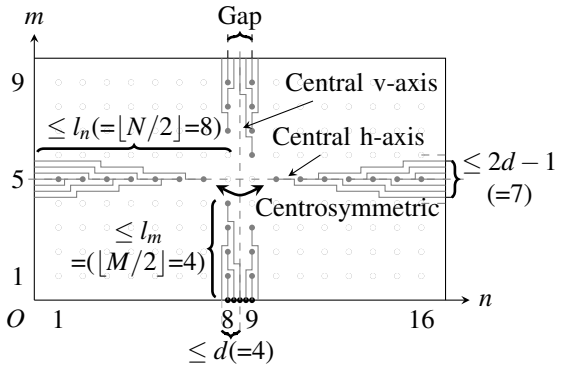


Fig. 11. Routing example of applying Rule-Central when N is odd and M is even: $N = 16$, $M = 9$, and $d = d_0 = 4$. The gap is between the 8^{th} and 9^{th} column.

in Line 9). When N is odd, Rule-2 is applied similarly.

When N is even, there are 2 columns closest to central v-axis. We first route no more than d internal terminals on the left-side column with routing channels to the left of central v-axis, and corresponding external terminals are as close to central v-axis as possible. Then we apply Rule-2 to route internal terminals on the right-side column to unrouted external terminals in the gap near central v-axis. As shown in Figure 11, $N = 16$ is even and $d = 4$, so no more than 4 internal terminals to the left-side of central v-axis are routed. Then Rule-2 is applied to route internal terminals along the right column towards unrouted external terminals to the right of central v-axis. The routing procedure continues until all external terminals in the gap (as marked in the figure) are routed. When M is even, Rule-2 is applied similarly.

Here, in Line 1-2, l_n, l_m is used to set the threshold on the number of internal terminals in a column/row to be routed, so that routing channels do not cross different subregions. The aim of applying Rule-2 with different values (0/1) of δ has two folds: (1) let external terminals closest to central v-axis/h-axis be routed first, and (2) when N or M is odd, terminal assignment rule is observed as stated in Section III-C. As stated above, only two set of internal terminals are routed: (1) those along the boundary between R_0 and R_1, and (2) those along the boundary between R_0 and R_2. For internal terminals along the boundary between R_1 and R_3, and along the boundary between R_0 and R_2, their routing solutions are obtained by transforming the coordinates (x, y) of routed channels by the following equation:

$$\begin{aligned} x' &= g_x - x \\ y' &= g_y - y \end{aligned} \tag{3}$$

where (x', y') is computed from (x, y). g_x and g_y are given in Equation (1).

2) Rule-General in R_0: The routing rule for general-state internal terminals (see Section III-D2), *Rule-General*, connects these internal terminals using both Rule-1 and Rule-2.

Definition 13 (Candidate External Terminal in R_0): The candidate external terminal in R_0 is defined as external terminal to be routed at current routing step in R_0. There are two candidate external terminals at each routing step corresponding to the two triangular regions, denoted as $T_x(t_x, 0)$ and $T_y(0, t_y)$. Assuming t'_x / t'_y denotes the minimum x / y coordinate of routed external terminal in lower/upper triangular region. Then $t_x = t'_x - 1$ / $t_y = t'_y - 1$.

Figure 12 shows t_x and t_y for candidate external terminals in an intermediate routing solution for R_0. As shown in the figure, when central-state internal terminals are routed, for routing general-state internal terminals, we start at the rightmost/topmost unoccupied external terminal and set it as candidate external terminal. Since external terminals for $t_x \in [16, 28]$ / $t_y \in [15, 22]$ are already occupied, the candidate external terminal for $t_x = 15$ / $t_y = 14$ is obtained. The candidate external terminals will be routed to certain unrouted general-state internal terminals. In the figure, the black lines denote routed channels for general-state internal terminals, and the bold black lines denote routed channels for central-state internal terminals.

Figure 12 shows an intermediate routing solution for general-state internal terminals in R_0, where N=13, M=12, d=d_0=4: the maximum column/row number of unrouted internal terminals in R_0 n_x=5 / n_y=5, candidate external terminals $T_x(15, 0)$ and $T_y(0, 14)$, candidate internal terminals in lower/upper triangular region $S_x[4, 2]/S_y[5, 5]$, and candidate routing channels in lower/upper triangular region c_x/c_y (marked in black dashed lines). The black lines denote routed channels for general-state internal terminals, and the bold black lines denote the routed channels for central-state internal terminals.

The intrinsic idea of Rule-General is to apply Rule-1 and Rule-2 for routing from candidate external terminal T_j to an unrouted internal terminal S_i in general state. In subregion R_0, each internal terminal is only connected to external terminals at the left half of x-axis or the bottom half of y-axis, i.e., each internal terminal is assigned to the lower/upper triangular region. For each round of routing, two candidate routing channels are obtained from T_x and T_y using Rule-1, which connect two different internal terminals in different triangular regions (see Figure 12). Then the shorter channel is first stored as a routing solution. For the longer one, only when certain conditions (cf. Line 22-29 in Algorithm 4 as explained later) are satisfied will it be stored as a routing solution. The merit of this routing method is that triangular regions are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2883908, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

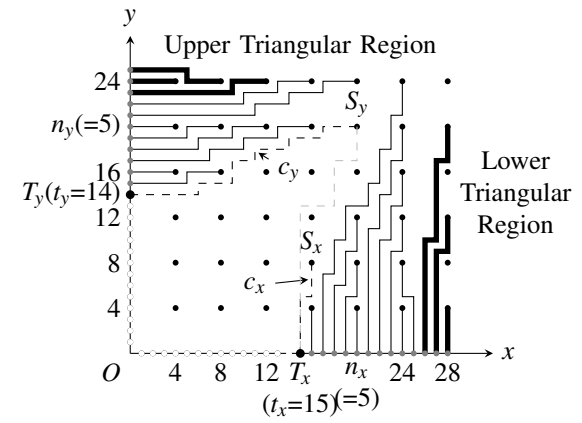


Fig. 12. Intermediate routing solution for general-state internal terminals in R_0.

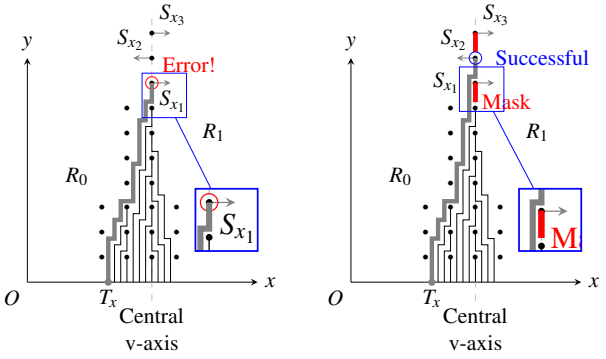


Fig. 13. Example of mask insertion: (a) without mask insertion, the computed channel in R_0 connects to $S_{x_1} \in R_1$, which violates terminal assignment rule, and (b) after masks insertion (denoted in red bars), the computed channel in R_0 connects to $S_{x_2} \in R_0$, which observes terminal assignment rule.

partitioned automatically during the routing process, which greatly enhances rule-based routing quality.

For certain general-state internal terminals near the corner of routing region or near the central-state internal terminals, the above routing method is not enough for completing the routing process. In this case, Rule-2 is applied to for obtaining routing solutions.

Algorithm 4 presents the rule-based routing method in subregion R_0. The input consists of subregion R_0 along with internal and external terminals to be routed. And the output is routed channels for R_0. In Line 1, we first obtain candidate external terminals as defined in Definition 13 and explained in Figure 12. Then coordinates of candidate external terminals are stored in $T_x(t_x,0)$ and $T_y(0,t_y)$. At the current step, the internal terminal to be routed is not determined yet. In Line 2, we obtain maximum column/row number n_x/n_y of unrouted internal terminals in R_0. In Line 3, we add *masks* (for marking a partial column/row of routing grids as obstacles) to guide routing process when N or M is an odd number. As shown in Figure 13(a), according to terminal assignment rule (5) in Section III-C, $S_{x_1}, S_{x_3} \in R_1$, and $S_{x_2} \in R_0$. However, when applying Rule-1 to route from an external terminal $(t_x,0)$ to an internal terminal on central v-axis, the connected internal terminal S_{x_1} will belong to R_0, which results in contradiction with terminal assignment rule. In Figure 13(b), masks (denoted by red bars) are inserted before applying Rule-1, such that connected internal terminal S_{x_2} will belong to R_0, which

Input: Subregion R_0 with general-state internal terminals and external terminals to be routed.
Output: Routed channels in R_0.

1 Obtain candidate external terminals $(t_x,0)$ and $(0,t_y)$;
2 Obtain maximum column/row number n_x/n_y of unrouted internal terminals in R_0.
3 Add masks to current state if N or M is odd;
4 **while** $n_x > 0 \wedge n_y > 0 \wedge t_x > 0 \wedge t_y > 0$ **do**
5 $f \leftarrow 0$;
6 **if** $t_x/t_y > \alpha$ **then**
7 $f \leftarrow 1$
8 **else if** $t_y/t_x > \alpha$ **then**
9 $f \leftarrow -1$
10 **if** $f \neq 1 \wedge n_y \times d \leq t_y$ **then**
11 Rule-2(Row, n_y, min$\{n_x,n_y\}$, δ=0);
12 Update t_y, n_x, n_y;
13 **Continue**;
14 **if** $f \neq -1 \wedge n_x \times d \leq t_x$ **then**
15 Rule-2(Column, n_x, min$\{n_x,n_y\}$, δ=0);
16 Update t_x, n_x, n_y;
17 **Continue**;
18 $S_x, c_x \leftarrow$Rule-1($(n_x \times d, M \times d)$, $(t_x,0)$);
19 $S_y, c_y \leftarrow$Rule-1($(N \times d, n_y \times d)$, $(0,t_y)$);
20 **if** $(f \neq 1 \wedge c_y = \varnothing) \vee (f \neq -1 \wedge c_x = \varnothing)$ **then**
21 **return** *Routing Failure;*
22 **if** $f = 1 \vee (f = 0 \wedge length(c_x) \leq length(c_y))$ **then**
23 Store c_x as routing solution;
24 **if** $f = 0 \wedge dist(S_y,(0,t_y)) \leq dist(S_y,(t_x,0))$ **then**
25 Store c_y as routing solution;
26 **else**
27 Store c_y as routing solution;
28 **if** $f = 0 \wedge dist(S_x,(t_x,0)) \leq dist(S_x,(0,t_y))$ **then**
29 Store c_x as routing solution;
30 Update t_x, t_y, n_x, n_y;
31 **if** $n_x > 0 \wedge n_y > 0 \wedge (t_x = 0 \vee t_y = 0)$ **then**
32 **return** *Routing Failure;*
33 Delete masks in current state if N or M is odd;
34 **return** *Routed channels*;

Algorithm 4: Rule-General in R_0.

resolves the contradiction with terminal assignment rule. The proposed terminal assignment rule is critical in determining overall routing quality. Experiments show that a contradiction with terminal assignment rule will result in significant routing failures. Specifically, when N is odd, masks will be inserted to $((N+1)/2)^{th}$ column for those unrouted internal terminals $S_i \notin R_0$. when M is odd, masks will be inserted similarly.

In Line 4-30, the while-loop is entered to route internal terminals in general state to available external terminals in subregion R_0 one by one. In Line 5-9, flag f is computed for evaluating the aspect ratio of routing subregion. For different aspect ratios, the routing methods are different. $f = 1$ denotes t_x is much larger than t_y, in which case only lower triangular region can be processed to reduce the difference between t_x and t_y. $f = -1$ denotes t_y is much larger than t_x, in which

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2883908, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

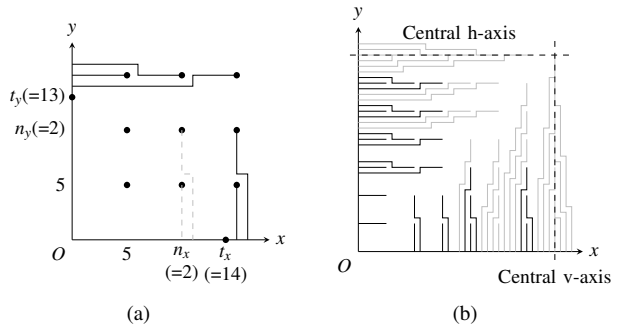


Fig. 14. Applying Rule-2 for general-state internal terminals: (a) as $n_x \times d \leq t_x$, t_x is to the right of column of n_x and the application condition of Rule-2 is satisfied; (b) routing solution for general-state internal terminals obtained by Rule-2, which is marked in black channels ($N = M = 13, d = d_0 = 5$).

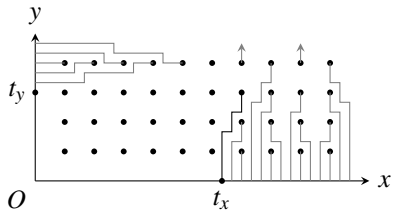


Fig. 15. Routing by Rule-General in R_0: as $t_x/t_y > \alpha$, $f = 1$ and only lower triangular region is processed ($N = 18$, $M = 7$, $d = d_0 = 3$).

case only the upper triangular region is processed to reduce the difference. $f = 0$ denotes that t_x is similar to t_y, and we can process these two triangular regions simultaneously. In Algorithm 4, $\alpha = 1.1$.

In Line 10-17, the application conditions of Rule-2 are checked. If $t_x \geq n_x \times d$ or $t_y \geq n_y \times d$, Rule-2 will be applied to obtain routing solution directly without obtaining candidate routing channels. Figure 14(a) shows an example where $t_x \geq n_x \times d$. The dashed lines in the figure denote the channels obtained by Rule-2. In Figure 14(b), the black lines denote all routing channels obtained by Rule-2 for general-state internal terminals.

Then in Line 18-19, Rule-1 is applied to obtain two routing channels (c_x, c_y) in two triangular regions along with corresponding internal terminals (S_x, S_y). The input to Rule-1 consists of pre-specified target internal terminals ($n_x \times d$,$M \times d$) and ($N \times d$,$n_y \times d$). Please note that ($n_x \times d$,$M \times d$) and ($N \times d$,$n_y \times d$) are used to provide routing guidance, i.e., to obtain the main and auxiliary routing directions. Rule-1 will return the first visited unoccupied internal terminal (S_x/S_y), which is possibly different from the given internal terminal. In Line 20-21, we check whether Rule-1 fails to obtain routing channels. If so, the internal terminals in general state fail to be routed and routing failure will be reported. More routing space between internal terminals is required in this case. In Line 22-29, different cases for the value of f is checked to determine whether c_x or c_y will be picked. These different cases are shown in Figure 12 and Figure 15. In Figure 12, t_x is similar to t_y, and the length of channel c_x is shorter than c_y, so we store c_x as routing solution. Then we check whether the length of c_y from S_y to t_y is shorter than that of S_y to t_x (denoted by gray dashed line). If so, it indicates that the current channel (c_y) is the optimal one, and will be stored as the routing solution. In Figure 15, as $t_x/t_y > \alpha$, the length-width aspect ratio is too high. Therefore, the value of f is 1, and only

Input: Number of internal terminal in horizontal/vertical direction N/M, and number of routing pitches d between adjacent internal terminals.
Output: Routing solution
1 Calculate g_x and g_y by Equation (1);
2 Partition routing region into 4 subregions by *terminal assignment rule*;
3 Call Algorithm 3 to route central-state internal terminals;
4 **for** $i \leftarrow 0$ *to* 3 **do**
5 **if** $i > 1$ *and* $|R_i| = |R_{3-i}|$ **then**
6 Copy channels from R_{3-i} to R_i by coordinate transformation;
7 **else**
8 Call Algorithm 4 to obtain routed channels in R_i;
9 **if** R_i *fails in routing* **then**
10 **return** routing failure;
11 **return** *Routing solution*;
Algorithm 5: Complete routing method.

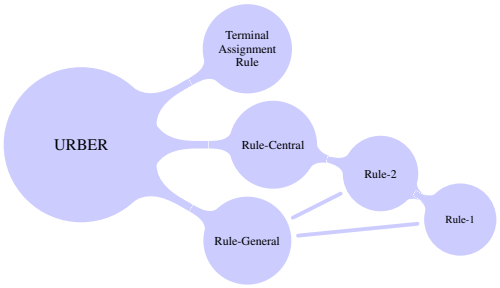


Fig. 16. Function-call relationship between all rule-based methods.

lower triangular region is processed to reduce the ratio. Here, $length(c)$ returns the total length of channel c, and $dist(A,B)$ returns the Manhattan distance between routing grids A and B. In Line 30, candidate external terminals T_x/T_y and column/row number n_x/n_y with maximum unrouted internal terminals are updated for next round.

In Line 31-32, we check the following condition: in one triangular region, there are unrouted internal terminals, and there are no unrouted external terminals. If so, routing failure will be reported. In this case, more routing grids between adjacent internal terminals are needed to successfully route all internal terminals. Otherwise, routing is successful and we will delete masks inserted in Line 3. Finally, all routed channels are returned as output.

G. Overall Routing Algorithm

Algorithm 5 presents the complete routing method. First, N, M, and d are obtained as input. Then g_x and g_y are computed by Equation (1), and terminal assignment rule is applied to divide the entire routing region into 4 subregions R_0, R_1, R_2, and R_3. Next, Rule-Central is applied to central-state internal terminals from all subregions, and Rule-General is applied to obtain routing solutions for general-state internal terminals in R_0. If any routing failure occurs, the value of d needs to be enlarged for successful routing. When routing solution for R_0 is obtained, according to the symmetric property of routing

Input: Number of internal terminals in horizontal/vertical direction N/M.
Output: Minimum number of routing pitches d_0 between adjacent internal terminals.

$L \leftarrow$ the lower bound of d_0;
$R \leftarrow$ the upper bound of d_0;
while $L < R$ **do**
  $d \leftarrow \lfloor (L+R)/2 \rfloor$;
  Call $URBER(N,M,d)$ in Algorithm 5;
  if *there is routing failure* **then**
    $L \leftarrow d+1$;
  else
    $R \leftarrow d$;
$d_0 \leftarrow L$;
return d_0;

Algorithm 6: Computation of d_0.

subregions (see Lemma 3), if $|R_0| = |R_3|$, we will reuse routing solution in R_0 and perform the coordinate transformation (i.e., flip by central axis) to obtain routing solution for R_3. The same routing process is conducted on remaining unrouted subregions. Finally, the routing solution of a given design is obtained. Figure 16 illustrates the function-call relationship between all rule-based methods.

H. Computation of d_0

We determine the optimal value d_0 of d by principle of the bisection. It is easy to understand that if d_i is feasible for successfully route all internal terminals, then $\forall d \geq d_i$ can be used for 100% routing completion. As each internal terminal corresponds to an external terminal, we have $2(g_x + g_y) \geq N \times M$, which is equivalent to $d_0 \geq \left\lfloor \frac{N \times M}{2(N+M+2)} \right\rfloor$. For upper bound of d, we know that all internal terminals can be successfully routed when $d = \lfloor \frac{1}{2}(\min\{N,M\}+1) \rfloor$. So we have $d_0 \leq \lfloor \frac{1}{2}(\min\{N,M\}+1) \rfloor$.

When lower and upper bounds of the optimal value d_0 are obtained, the principle of the bisection can be applied for computing d_0 for minimizing routing area. At each step of the bisection, URBER(N,M,d) is applied to check whether d is feasible for successfully routing the given design. If there is routing failure, the value of d needs to be enlarged, and the lower bound will be increased. Otherwise, $d \geq d_0$ and the upper bound will be decreased. When the lower bound and the upper bound is equal, we obtain the value of d as d_0.

Algorithm 6 presents the details of how d_0 is computed. According to the special property of *terminal assignment rule* in Section III-C, before coordinate transformation, we have $|R_0| \geq |R_3|$ and $|R_1| \geq |R_2|$. Therefore, when the value of d can be used to successfully route R_0 (R_1), the same d can also be used for R_3 (R_2). As a result, only subregion R_0 and R_1 are needed for computing d_0 when calling rule-based routing algorithm (URBER).

I. Approximation Formula of d_0

To speed up the computation of d_0 in Algorithm 6, we propose an empirical formula for approximating the value of d_0 as

$$f(N,M) = \frac{\alpha \times N \times M}{N+M-\beta} - \gamma \tag{4}$$

where $\alpha = 0.586$, $\beta = 5.236$ and $\gamma = 2.256$. We have conducted experiments for all benchmarks with $3 \leq N,M \leq 600$, from which the error between d_0 and $f(N,M)$ is less than 2.7 routing grids (see Figure 19 for the accuracy on certain values of N and M). Although parameters α, β, and γ are obtained by fitting to $3 \leq N,M \leq 600$, Equation (4) works well for benchmarks of much larger scale (See Table III). From the equation, we have $d = O(\frac{NM}{N+M})$, which is adopted in following complexity analysis.

J. Complexity Analysis

Theorem 1: Given the numbers of internal terminals in horizontal and vertical directions, N and M, respectively, URBER runs in $O(\frac{N^3M^3}{(N+M)^2})$ time.

Proof: Assuming $length(c)$ denotes the length of channel c, c_c denotes a channel for a central-state internal terminal, c_g denotes a channel for a general-state internal terminal. It takes $O(1)$ time for applying terminal assignment rule on each of the $N \times M$ internal terminals. So it takes $O(NM)$ for the assignment of all internal terminals. For applying Rule-Central, it takes $O(\sum length(c_c))$ to calculate all channels connecting central-state internal terminals. For applying Rule-General, it takes $O(\sum length(c_g))$ to calculate all channels connecting general-state internal terminals. As a result, computing all channels by rule-based routing takes $O(\sum length(c_c) + \sum length(c_g))$. On one hand, there are no overlaps or crossings between channels. On the other hand, for most benchmarks in the experiments, about 72% of total routing grids in the routing region are occupied. Therefore, we infer that $O(\sum length(c_c) + \sum length(c_g))$ be equivalent to the total number of grids in routing region $O(g_x g_y)$. Moreover, the coordinate transformation performed on a routing subregion also takes $O(g_x g_y)$ time. According to Equation (1), $O(g_x g_y) = O(NMd^2)$. And according to Equation (4), $d = O(\frac{NM}{N+M})$. So we have $O(g_x g_y) = O(\frac{N^3M^3}{(N+M)^2})$. Therefore, the overall runtime complexity of the proposed routing algorithm is $O(\frac{N^3M^3}{(N+M)^2})$. ■

Theorem 2: Let V denote the total number of internal terminals ($V = NM$), and let $k = \max\{N,M\}/\min\{N,M\} \geq 1$ denote the aspect ratio of given rectangular design. URBER runs in $O(\frac{V^2}{k})$ time.

Proof: Without loss of generality, assume $N \geq M$. Then we have $N = \sqrt{Vk}$ and $M = \sqrt{V/k}$. Therefore, the overall complexity is equal to $O(\frac{N^3M^3}{(N+M)^2}) = O(\frac{V^2}{k+2+1/k}) = O(\frac{V^2}{k})$. ■

For rectangular-shape designs ($k > 1$), we have $1/k < 1$. In contrast, for square-shaped designs, we have $1/k = 1$. Therefore, according to the runtime complexity $O(\frac{V^2}{k})$, URBER runs faster for rectangular-shape designs than for square-shaped designs. Experimental results in Table III verify above analysis.

IV. Experimental Results

We have implemented URBER in C++ programming language. Our system is tested on an E5620 2.40GHz Intel

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2883908, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
EXPERIMENTAL RESULTS ON SMALL-SCALE BENCHMARKS.

$N \times M$	d_0	$g_x \times g_y$	Total Length				Error(%)			CPU(s)				Speedup of URBER		
			MCF	MaxF	Fr41	URBER	MaxF	Fr41	URBER	MCF	MaxF	Fr41	URBER	MCF	MaxF	Fr41
$30 \times 30 = 900$	9	279×279	**55112**	55196	55188	**55112**	0.152	0.138	**0**	27.65	1.33	5.79	**0.019**	1.46E+03	7.00E+01	3.05E+02
$45 \times 45 = 2025$	14	644×644	**273183**	273811	274583	**273183**	0.230	0.512	**0**	554.43	16.59	1704.61	**0.025**	2.22E+04	6.64E+02	6.82E+04
$55 \times 55 = 3025$	17	952×952	**602856**	603975	603280	**602856**	0.186	0.070	**0**	2712.88	50.08	2202.80	**0.032**	8.48E+04	1.57E+03	6.88E+04
$64 \times 64 = 4096$	19	1235×1235	**1092600**	1093556	1099416	**1092600**	0.086	0.624	**0**	6953.62	133.24	12777.88	**0.037**	1.88E+05	3.60E+03	3.45E+05
$71 \times 71 = 5041$	22	1584×1584	**1657902**	1660468	1660358	**1657902**	0.155	0.148	**0**	18402.45	234.55	61134.90	**0.053**	3.47E+05	4.43E+03	1.15E+06
$72 \times 13 = 936$	6	438×84	**26498**	26858	26502	**26498**	1.359	0.015	**0**	11.03	0.18	2.81	**0.018**	6.13E+02	1.00E+01	1.56E+02
$77 \times 26 = 2002$	11	858×297	**183686**	184642	184776	**183686**	0.520	0.593	**0**	319.15	6.12	555.45	**0.022**	1.45E+04	2.78E+02	2.52E+04
$111 \times 27 = 2997$	12	1344×336	**326743**	328523	326939	**326743**	0.545	0.060	**0**	932.46	12.67	15.55	**0.026**	3.59E+04	4.87E+02	5.98E+02
$69 \times 58 = 4002$	19	1330×1121	**1034338**	1035776	1034972	**1034338**	0.139	0.061	**0**	6901.52	112.77	22223.01	**0.035**	1.97E+05	3.22E+03	6.35E+05
$98 \times 51 = 4998$	20	1980×1040	**1399334**	1403899	1400562	1399338	0.326	0.088	**0.0003**	12352.41	152.00	45184.25	**0.038**	3.25E+05	4.00E+03	1.19E+06
Avg.	-	-	-	-	-	-	0.370	0.231	**0**	-	-	-	-	1.22E+05	1.83E+03	3.48E+05

TABLE III
EXPERIMENTAL RESULTS ON LARGE-SCALE BENCHMARKS.

$N \times M$	d_0	$g_x \times g_y$	$\lvert f(N,M) - d_0 \rvert$	Total Length	CPU(s)	Memory(M)
$316 \times 316 = 99856$	93	29481×29481	1.8945	629985640	7.50	348.8
$447 \times 447 = 199809$	132	59136×59136	2.5134	2517494576	42.42	964.5
$548 \times 548 = 300304$	161	88389×88389	1.9212	5679142936	65.10	1749
$632 \times 632 = 399424$	186	117738×117738	2.3097	10042278616	114.49	2861
$707 \times 707 = 499849$	208	147264×147264	2.3351	15720245427	256.54	4155
$375 \times 267 = 100125$	92	34592×24656	2.1131	608817645	8.94	341.8
$532 \times 376 = 200032$	129	68757×48633	1.4117	2420475700	24.73	946.5
$858 \times 350 = 300300$	141	121119×49491	3.0535	4385731204	38.23	1532
$904 \times 442 = 399568$	171	154755×75753	1.3809	8477760224	76.60	2572
$949 \times 527 = 500123$	196	186200×103488	1.0092	14012963087	189.95	3868

Xeon Linux server with 16 cores and 40GB memory. Only a single thread is used. The routing results are obtained with the given value of d_0, and the principle of the bisection is not applied here. In the overall flow, approximation approach for d_0 (see Section III-I) can be used instead. Table II shows routing results on small-scale cases, which already cost much runtime for the network-flow-based method. In Table II, (N,M) gives the number of internal terminals in horizontal and vertical directions, respectively, "d_0" gives the minimum number of routing pitches between adjacent internal terminals for successfully routing the given design, "(g_x, g_y)" gives total number of routing grids in horizontal and vertical directions, respectively, "Total Length" gives total length of routed channels, "MCF" gives the optimal routing results using min-cost flow-based formulation, "MaxF" gives the sub-optimal routing results using maximum network-flow formulation[1], "Fr41" is a global routing method proposed in [44], and "URBER" gives our rule-based routing results. We have tried to use FastRoute 4.1 to run on single-layer routing. However, the program always outputs "heap underflow" error. Thus, we could only apply FastRoute 4.1 with two-layer routing setup, i.e., the first layer is set for horizontal routing, and the second layer is set for vertical routing, with routing capacity set to be 1. Since the two-layer routing setup is adopted in FastRoute 4.1, the routing problem is easier for "Fr41".

In the first 5 benchmarks, $N = M$, and in the last 5 benchmarks, $N \neq M$ (randomly synthesized). Different benchmarks cover the scale from 1k to 5k. From column under "Total Length", URBER obtains near-optimal routing solution. In fact, there are very few benchmarks for which our routing method cannot obtain optimal routing solutions. The column under "Error" gives the error in percentage. From results, the maximum network-flow based method and the global routing method obtains significantly degraded routing solutions than URBER. From column under "CPU(s)", which gives the runtime of different methods, URBER is order of magnitudes faster than network-flow based method and global routing method. URBER successfully routes all small-scale benchmarks within 0.1 seconds, which take up to 5 hours for the min-cost flow-based method and 17 hours for the global routing method. Moreover, most of the routing results by our method are optimal. The column under "Speedup of URBER" gives the speedup of URBER over existing methods. The MCF/MaxF/Fr41 speedup is computed as (MCF's/MaxF's/Fr41's CPU time)/(URBER's CPU time). From results, the network-flow-based method and global-routing-based method do not scale well for the sizes of benchmarks. In contrast, URBER scales well to large problem size. Therefore, the larger the problem size, the more significant speedup URBER obtains. Moreover, in all experiments, the maximum memory consumption of URBER is less than 4.2GB, which is considerably small for modern servers. We observe that even with similar scale of total routing grids, as the value of $k = N/M$ is greater than 1 for rectangular routing region ($N \neq M$) and equal to 1 for square region ($N = M$), our routing method runs faster for rectangular routing regions according to the complexity $O(\frac{V^2}{k})$ (see Section III-J). The runtime in Table II confirms the above analysis.

[1]Similar to [43], MCF is the network-flow formulation on fine routing grids, which guarantees to route optimality. And MaxF is similar to MCF but without the edge cost for minimized total length.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2883908, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

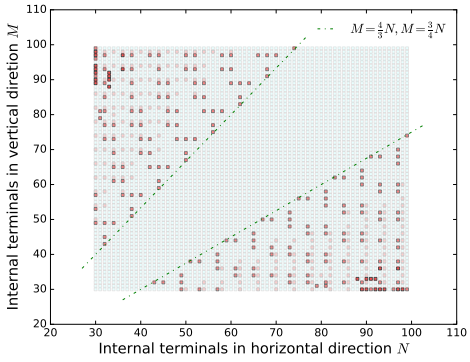


Fig. 17. Routing solutions of URBER.

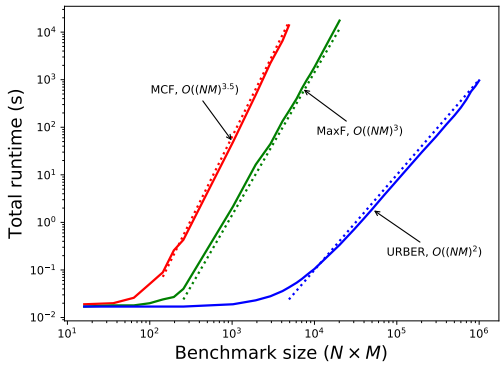


Fig. 18. Runtime of different methods in log scale (N=M).

Table III shows the routing results on large-scale benchmarks, where the number of internal terminals ranges from 100k to 500k. For large-scale benchmarks, even using the maximum network-flow method, it is not possible to obtain feasible routing solutions in acceptable runtime. In contrast, using our routing method without any hierarchical or multilevel speedup techniques, the routing solutions can be successfully obtained within 5 minutes. All routing solutions have passed validity check. From the table, the maximum memory consumption is less than 4.2 GB. The column under "$|f(N,M)-d_0|$" gives absolute difference between actual value of d_0 and estimated value by approximation formula in Equation (4). Although the fitted parameters in Equation (4) are obtained for benchmarks with $N,M \leq 600$, the maximum error for benchmarks with N or $M > 600$ is only about 3 routing grids, which still demonstrates high accuracy.

We have tested all benchmarks for $30 \leq N,M \leq 100$ (totally 5,041 benchmarks for either $N = M$ or $N \neq M$). When $30 \leq N,M \leq 100$, routing error of URBER is less than 0.005%. Figure 17 shows all routing solutions of these benchmarks, where optimal routing solutions are denoted in light-blue dots, and sub-optimal routing solutions are denoted in red dots. The darker the red dot, the worse the routing solution. Specifically, when $\frac{M}{N} \in (\frac{3}{4}, \frac{4}{3})$, optimal routing solutions are always obtained, which have been verified by the min-cost flow-based method. Among all benchmarks, only about 408 ($\sim$8.1%) cases are routed with the sub-optimal solution. The maximum error in channel length for URBER is 20, which is obtained when $N = 92$, $M = 34$, and $d_0 = 15$. Considering the large scale of this benchmark, i.e., total 734,296 routing grids, the error of 20 routing grids is negligible.

Figure 18 shows the runtime of different methods on square-shape benchmarks with $N = M$. For better readability, x- and y-axis are drawn in log scale. From the figure, the min-cost flow-based method runs very slow, and the runtime increases dramatically when the problem size is large. The maximum network-flow method runs a bit faster, but with significantly degraded routing quality. In contrast, URBER runs orders of magnitude faster with high-quality routing solutions. Specifically, when $N \times M$ is larger than 10^4, the runtime of URBER fits well to $O((NM)^2)$ ($10^{-9}(NM)^2$) line denoted in blue dots, which validates the runtime complexity analysis that the proposed rule-based routing method runs in $O(\frac{N^3M^3}{(N+M)^2})$.

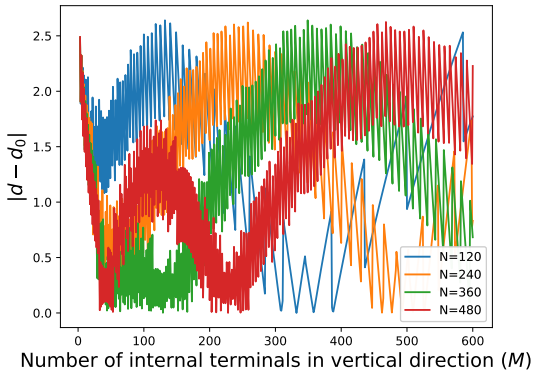


Fig. 19. Prediction errors of approximation formula (Equation (4)) over actual values of d_0.

Figure 19 shows the prediction errors between approximation formula ($f(N,M)$ in Equation (4)) and actual values of d_0. For different N and M values, the approximation equation of d_0 well predicts the actual values. The maximum error for these different cases ($N = 120, 240, 360, 480$ and $M \in [3, 600]$) is 2.64 routing grids, which proves the accuracy of the proposed approximation formula. Please note that the approximation formula can be used to significantly reduces computation overhead in the principle of the bisection for determining the actual value of d_0.

Figure 20 shows the routing results for the benchmark with $N = 72$ and $M = 13$ (6^{th} benchmark in Table II) obtained by three different methods. Figure 20(a), Figure 20(b), Figure 20(c), and Figure 20(d) give the routing results of the min-cost flow-based method (MCF), the maximum network-flow method (MaxF), the global routing method (Fr41) and URBER, respectively. Figure 21 shows the routing results for the benchmark with $N = M = 30$ (1^{st} benchmark in Table II) obtained by MCF and our method, respectively. From these figures, URBER obtains much more regular routing solutions than the network-flow-based and global routing methods. We anticipate this as a critical property for certain biochip applications, where we expect similar outputs at symmetric terminals. An online demonstrative system of URBER can be found at http://biocad.cs.tsinghua.edu.cn/urber.html.

V. Conclusion

We have proposed the first ultrafast rule-based escape routing method, called URBER, for the drug sample delivery biochip architecture for microwell array-based drug screening applications. Experimental results have shown that URBER

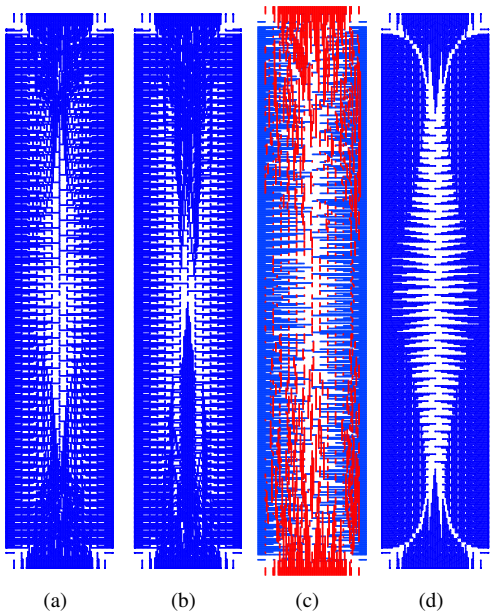


Fig. 20. Routing results of different methods for benchmark with $N = 72$ and $M = 13$: (a) MCF, (b) MaxF, (c) Fr41, and (d) URBER. Fr41 is drawn by two colors due to two-layer routing setup, with many cross points between different wires on two layers.

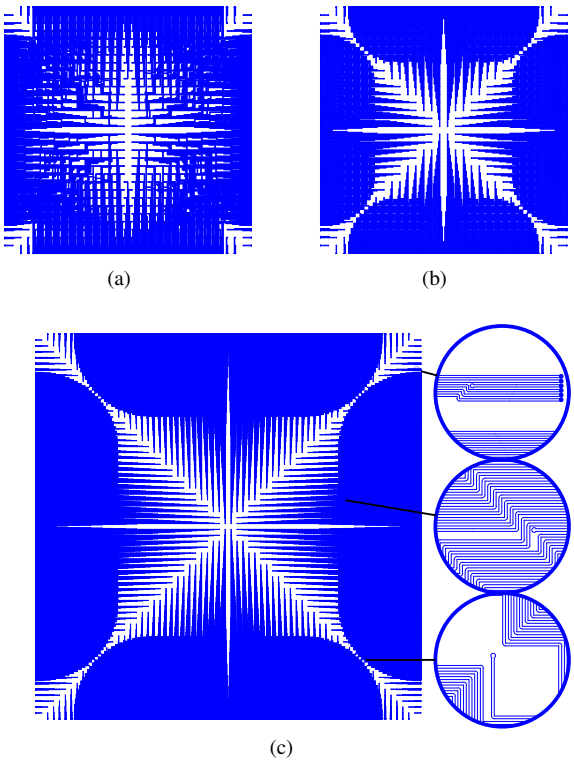


Fig. 21. Routing results of different methods: (a) CostFlow for $N = M = 30$, (b) Our routing method for $N = M = 30$, and (c) Our routing method for $N = M = 64$.

successfully addresses the critical large-scale routing challenges, and scales very well in both runtime and memory for large-scale benchmarks, which cannot be solved by existing min-cost flow-based method in acceptable runtime. Specifically, URBER obtains high-quality routing solutions for 100k microwell array only within minutes, confirming the effective and efficient routing capacity of the proposed rule-based routing method. Future work includes extending URBER to process diagonal and missing-terminal escape routing problems, as well as theoretical analysis of the solution quality.

REFERENCES

[1] J. Gole, A. Gore, A. Richards, Y.-J. Chiu, H.-L. Fung, D. Bushman, H.-I. Chiang, J. Chun, Y.-H. Lo, and K. Zhang, "Massively parallel polymerase cloning and genome sequencing of single cells using nanoliter microwells," *Nature Biotechnology*, vol. 31, no. 12, p. 1126, 2013.

[2] R. E. Assal, U. A. Gurkan, P. Chen, F. Juillard, A. Tocchio, T. Chinnasamy, C. Beauchemin, S. Unluisler, S. Canikyan, A. Holman, S. Srivatsa, K. M. Kaye, and U. Demirci, "3-d microwell array system for culturing virus infected tumor cells," *Scientific Reports*, vol. 6, p. 39144, 2016.

[3] P. Zhang, J. Zhang, S. Bian, Z. Chen, Y. Hu, R. Hu, J. Li, Y. Cheng, X. Zhang, Y. Zhou, X. Chen, and P. Liu, "High-throughput superhydrophobic microwell arrays for investigating multifactorial stem cell niches," *Lab on a Chip*, vol. 16, no. 16, pp. 2996–3006, 2016.

[4] T.-Y. Tu, Z. Wang, J. Bai, W. Sun, W. K. Peng, R. Y.-J. Huang, J.-P. Thiery, and R. D. Kamm, "Rapid prototyping of concave microwells for the formation of 3d multicellular cancer aggregates for drug screening," *Advanced Healthcare Materials*, vol. 3, no. 4, pp. 609–616, 2014.

[5] W. H. Minhass, P. Pop, and J. Madsen, "System-level modeling and synthesis of flow-based microfluidic biochips," *Proc. of CASES*, 2011, pp. 225–234.

[6] W. H. Minhass, P. Pop, and J. Madsen, "Synthesis of biochemical applications on flow-based microfluidic biochips using constraint programming," *Proc. of DTIP*, 2012, pp. 37–41.

[7] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga, "Architectural synthesis of flow-based microfluidic large-scale integration biochips," *Proc. of CASES*, 2012, pp. 181–190.

[8] T.-M. Tseng, B. Li, T.-Y. Ho, and U. Schlichtmann, "Reliability-aware synthesis for flow-based microfluidic biochips by dynamic-device mapping," *Proc. of DAC*, 2015, p. 141.

[9] T.-M. Tseng, B. Li, M. Li, T.-Y. Ho, and U. Schlichtmann, "Reliability-aware synthesis with dynamic device mapping and fluid routing for flow-based microfluidic biochips," *IEEE Trans. on CAD*, vol. 35, no. 12, pp. 1981–1994, 2016.

[10] J. McDaniel, B. Parker, and P. Brisk, "Simulated annealing-based placement for microfluidic large scale integration (mlsi) chips," *Proc. of VLSI-SoC*, 2014, pp. 1–6.

[11] C.-X. Lin, C.-H. Liu, I.-C. Chen, D. T. Lee, and T.-Y. Ho, "An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips," *Proc. of DAC*, 2014, pp. 1–6.

[12] A. Grimmer, Q. Wang, H. Yao, T.-Y. Ho, and R. Wille, "Close-to-optimal placement and routing for continuous-flow microfluidic biochips," *Proc. of ASP-DAC*, 2017, pp. 530–535.

[13] N. Amin, W. Thies, and S. Amarasinghe, "Computer-aided design for microfluidic chips based on multilayer soft lithography," *Proc. of ICCD*, 2009, pp. 2–9.

[14] W. H. Minhass, P. Pop, J. Madsen, and T.-Y. Ho, "Control synthesis for the flow-based microfluidic large-scale integration biochips," *Proc. of ASP-DAC*, 2013, pp. 205–212.

[15] K.-H. Tseng, S.-C. You, J.-Y. Liou, and T.-Y. Ho, "A top-down synthesis methodology for flow-based microfluidic biochips considering valve-switching minimization," *Proc. of ISPD*, 2013, pp. 123–129.

[16] K. Hu, T.-A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer optimization for flow-based mvlsi microfluidic biochips," *Proc. of CASES*, 2014, pp. 1–10.

[17] H. Yao, T.-Y. Ho, and Y. Cai, "Pacor: practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips," *Proc. of DAC*, 2015, pp. 142:1–142:6.

[18] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer routing and control-pin minimization for flow-based microfluidic biochips," *IEEE Trans. on CAD*, vol. 36, no. 1, pp. 55–68, 2017.

[19] Q. Wang, Y. Xu, S. Zuo, H. Yao, T.-Y. Ho, B. Li, U. Schlichtmann, and Y. Cai, "Pressure-Aware Control Layer Optimization for Flow-Based Microfluidic Biochips," *IEEE TBioCAS*, vol. 11, no. 6, pp. 1488–1499, 2017.

[20] Q. Wang, S. Zuo, H. Yao, T.-Y. Ho, B. Li, U. Schlichtmann, and Y. Cai, "Hamming-distance-based valve-switching optimization for control-layer multiplexing in flow-based microfluidic biochips," *Proc. of ASP-DAC*, 2017, pp. 524–529.

[21] H. Yao, Q. Wang, Y. Ru, Y. Cai, and T. Ho, "Integrated flow-control codesign methodology for flow-based microfluidic biochips," *IEEE Design & Test*, vol. 32, no. 6, pp. 60–68, 2015.

[22] T.-M. Tseng, M. Li, B. Li, T.-Y. Ho, and U. Schlichtmann, "Columba: co-layout synthesis for continuous-flow microfluidic biochips," *Proc. of DAC*, 2016, pp. 1–6.

[23] Q. Wang, H. Zou, H. Yao, T.-Y. Ho, R. Wille, and Y. Cai, "Physical Co-Design of Flow and Control Layers for Flow-Based Microfluidic Biochips," *IEEE Trans. on CAD*, vol. 37, no. 6, pp. 1157–1170, 2018.

[24] T.-M. Tseng, M. Li, D. N. Freitas, T. McAuley, B. Li, T.-Y. Ho, I. E. Araci, and U. Schlichtmann, "Columba 2.0: A Co-Layout Synthesis Tool for Continuous-Flow Microfluidic Biochips," *IEEE Trans. on CAD*, vol. 37, no. 8, pp. 1588–1601, 2018.

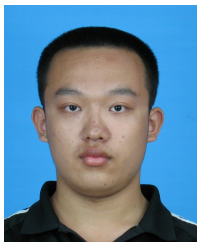
[25] T.-M. Tseng, M. Li, D. N. Freitas, A. Mongersun, I. E. Araci, T.-Y. Ho, and U. Schlichtmann, "Columba S: a scalable co-layout design automation tool for microfluidic large-scale integration," *Proc. of DAC*, 2018, pp. 163:1–163:6.

[26] T.-M. Tseng, B. Li, U. Schlichtmann, and T.-Y. Ho, "Storage and caching: Synthesis of flow-based microfluidic biochips," *IEEE Design & Test*, vol. 32, no. 6, pp. 69–75, 2015.

[27] C. Liu, B. Li, H. Yao, P. Pop, T.-Y. Ho, and U. Schlichtmann, "Transport or store?: Synthesizing flow-based microfluidic biochips using distributed channel storage," *Proc. of DAC*, 2017, pp. 1–6.
[28] I. E. Araci, P. Pop, and K. Chakrabarty, "Microfluidic very large-scale integration for biochips: Technology, testing and fault-tolerant design," *Proc. of ETS*, 2015, pp. 1–8.
[29] P. Pop, I. E. Araci, and K. Chakrabarty. "Continuous-flow biochips: Technology, physical-design methods, and testing," *IEEE Design & Test*, vol. 32, no. 6, pp. 8–19, 2015.
[30] C. Liu, B. Li, B. B. Bhattacharya, K. Chakrabarty, T.-Y. Ho, and U. Schlichtmann, "Testing microfluidic fully programmable valve arrays (fpvas)," *Proc. of DATE*, 2017, pp. 91–96.
[31] C. Liu, B. Li, T.-Y. Ho, K. Chakrabarty, and U. Schlichtmann, "Design-for-testability for continuous-flow microfluidic biochips," *Proc. of DAC*, 2018, pp. 164:1–164:6.
[32] J. Wang, P. Brisk, and W. H. Grover, "Random design of microfluidics," *Lab on a Chip*, vol. 16, pp. 4212–4219, 2016.
[33] W. Ji, T.-Y. Ho, and H. Yao, "More Effective Randomly-Designed Microfluidics," *Proc. of ISVLSI*, 2018, pp. 660–665.
[34] H. Kong, T. Yan, M. D. F. Wong, and M. M. Ozdal, "Optimal bus sequencing for escape routing in dense pcbs," *Proc. of ICCAD*, 2007, pp. 390–395.
[35] Y.-J. Lee, H.-M. Chen, and C.-Y. Chin, "On simultaneous escape routing of length matching differential signalings," *Proc. of EDAPS*, 2013, pp. 177–180.
[36] S.-I. Lei and W.-K. Mak, "Simultaneous constrained pin assignment and escape routing considering differential pairs for fpga-pcb co-design," *IEEE Trans. on CAD*, vol. 32, no. 12, pp. 1866–1878, 2013.
[37] S.-I. Lei and W.-K. Mak, "Optimizing pin assignment and escape routing for blind-via-based pcbs," *IEEE Trans. on CAD*, vol. 35, no. 2, pp. 246–259, 2016.
[38] T. Yan and M. D. F. Wong, "Correctly modeling the diagonal capacity in escape routing," *IEEE Trans. on CAD*, vol. 31, no. 2, pp. 285–293, 2012.
[39] S. K. W. Dertinger, D. T. Chiu, N. L. Jeon, and G. M. Whitesides, "Generation of gradients having complex shapes using microfluidic networks," *Analytical Chemistry*, vol. 73, no. 6, pp. 1240–1246, 2001.
[40] N. L. Jeon, S. K. W. Dertinger, D. T. Chiu, I. S. Choi, A. D. Stroock, and G. M. Whitesides. "Generation of solution and surface gradients using microfluidic systems," *Langmuir*, vol. 16, no. 22, pp. 8311–8316, 2000.
[41] H. Xiang, X. Tang, and M. D. F. Wong, "Min-cost flow-based algorithm for simultaneous pin assignment and routing," *IEEE Trans. on CAD*, vol. 22, no. 7, pp. 870–878, 2003.
[42] J.-W. Fang, I.-J. Lin, Y.-W. Chang, and J.-H. Wang, "A network-flow-based rdl routing algorithmz for flip-chip design," *IEEE Trans. on CAD*, vol. 26, no. 8, pp. 1417–1429, 2007.
[43] Q. Wang, Z. Li, H. Cheong, O.-S. Kwon, H. Yao, T.-Y. Ho, K. Shin, B. Li, U. Schlichtmann, and Y. Cai, "Control-fluidic codesign for paper-based digital microfluidic biochips," *Proc. of ICCAD*, 2016, pp. 1–8.
[44] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: global router with efficient via minimization," *Proc. of ASP-DAC*, 2009, pp. 576–581.

Jiayi Weng is currently an undergraduate student in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He will receive the B.S. degree in 2020.

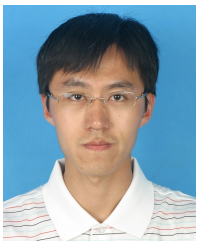
Tsung-Yi Ho (SM'12) received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. He is a Professor with the Department of Computer Science of National Tsing Hua University, Hsinchu, Taiwan. From 2007 to 2014 and 2015, he was with National Cheng Kung University and National Chiao Tung University, respectively. His research interests include design automation and test for microfluidic biochips and nanometer integrated circuits. He has presented 10 tutorials and contributed 10 special sessions in ACM/IEEE conferences, all in design automation for microfluidic biochips. He was a recipient of the Best Paper Awards at the VLSI Test Symposium (VTS) in 2013 and IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. He currently serves as an ACM Distinguished Speaker, a Distinguished Lecturer of the IEEE CAS Society, the Chair of the IEEE Computer Society Tainan Chapter, the Chair of the ACM SIGDA Taiwan Chapter, and Associate Editor of the ACM Journal on Emerging Technologies in Computing Systems, ACM Transactions on Design Automation of Electronic Systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Very Large Scale Integration Systems, Guest Editor of IEEE Design & Test of Computers.



Weiqing Ji is currently a Ph.D. candidate student in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include design automation methods of microfluidic biochips.

Peng Liu received both his B.Eng. degree in Environmental Engineering and M.S. degree in Biochemistry and Molecular Biology from Tsinghua University. He then graduated from the University of California, Berkeley with a Ph.D. in Bioengineering. He completed his postdoctoral training at Sandia National Laboratories, US. Peng Liu joined the Department of Biomedical Engineering, Tsinghua University School of Medicine, as a principal investigator and an associate professor in 2012. Currently, Dr. Liu's research interests include: 1) developing fully integrated microfluidic systems for point-of-care diagnosis, forensic short tandem repeat analysis, etc; 2) developing high-throughput cell microarray platforms for cell manipulation, culture, and analysis.

Mengdi Bao received her B.S. degree in 2015 at Southern Medical University, China and completed her M.S. degree in 2017 with Prof. Kazunori Hoshino at the University of Connecticut. After that, she joined Prof. Hailong Yao's group at Tsinghua University and worked on developing microdevices and immunoassay as a research assistant. She is now a R&D engineer in Berry Genomics Company working on process optimization for identification hydroxymethylcytosine modifications in cell-free DNA to diagnose cancer. Her research focuses on CRISPR based nanosensors.



Hailong Yao (SM'15) received the B.S. degree in computer science and technology from Tianjin University, Tianjin, China, in 2002, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2007. From 2007 to 2009, he was a postdoctoral research scholar in the Department of Computer Science and Engineering, University of California at San Diego, La Jolla. He joined the Department of Computer Science and Technology at Tsinghua University as an assistant professor in September 2009. His research interests include computer-aided design for microfluidic biochips and very large scale integration (VLSI) physical design. Dr. Yao received two Best Paper Award Nominations at ICCAD in 2006 and 2008, respectively. He received the ISQED Best Paper Award Nomination in 2011 and received the SASIMI Best Paper Award in 2016.