Optimization of Fluid Loading on Programmable Microfluidic Devices for Bio-protocol Execution

Satoru Maruyama*, Debraj Kundu†, Shigeru Yamashita* and Sudip Roy†
*College of Information Science and Engineering, Ritsumeikan University, Japan
†CoDA Laboratory, Department of Computer Science and Engineering, IIT Roorkee, India
Email-IDs: {inari@ngc.is.ritsumei.ac.jp, debraj0knd@gmail.com, ger@cs.ritsumei.ac.jp, sudiproy.fcs@iitr.ac.in}

Abstract— Recently, *Programmable Microfluidic Device* (PMD) has got an attention of the design automation communities as a new type of microfluidic biochips. For the design of PMD chips, one of the important tasks is to minimize the number of flows for loading the reactant fluids into specific cells (by creating some flows of the fluids) before the bio-protocol is executed. Nevertheless of the importance of the problem, there has been almost no work to study this problem. Thus, in this paper, we intensively study this fluid loading problem in PMD chips. First, we successfully formulate the problem as a constraint satisfaction problem (CSP) to solve the problem optimally for the first time. Then, we also propose an efficient heuristic called *Determining Flows from the Last (DFL)* method for larger problem instances. *DFL* is based on a novel idea that it is better to determine the flows from the last flow unlike the state-of-the-art method *Fluid Loading Algorithm for PMD* (*FLAP*) [Gupta *et al.*, TODAES, 2019]. Simulation results confirm that the exact method can find the optimal solutions for practical test cases, whereas our heuristic can find near-optimal solutions, which are better than those obtained by *FLAP*.

I. Introduction

An advanced architecture of continuous flow microfluidic biochips (CMFBs) [1] is experimentally demonstrated by Fidalgo *et al.* [2] known as *programmable microfluidic device* (PMD), which is a fully software programmable chip. In order to execute a bio-protocol on a PMD chip, first, the desired amount of different biochemical fluids are to be loaded in the PMD cells. Then the subsequent operations (mix, split, separate, perfuse, detect, etc.) are performed on that PMD chip following the sequence of steps (denoted by a sequencing graph [3]) indicated by the bio-protocol. Till now, a few design methods have been reported in literature specific for only PMD chips.

A recent work by Gupta *et al.* [4] proposed a dilution algorithm for sample preparation using PMD chips and the mapping (module binding) of the dilution graph (a kind of sequencing graph with mix-split steps [5]) into a PMD chip. Then, it tuned out that, for PMD chips one of the important tasks is to load reactant fluids (reagents) into specific cells before a bio-protocol is executed. In order to load different reagents in the PMD chips with only one input port, we need to create the *flows of those fluids* one-by-one. If the total number of such fluid-flows increases, then we will need more amount of expensive reagents, which incurs extra cost of the biochip. Hence, the number of flows required to load all the reagents into different cells of a PMD chip is an important factor to minimize. The minimization of this number of fluid-flows along with determining the flow paths is referred to as the *fluid loading problem*. To the best of our knowledge, the state-of-the-art method for fluid loading problem is *FLAP* [4]. Since the problem is *NP-hard*, *FLAP* is a heuristic based on an idea that one flow can always load a reagent into the "L-shaped" connected cells of a PMD chip.

In this paper, we study the *fluid loading problem* intensively. First, we successfully formulate the problem as a constraint satisfaction problem (CSP) [6] to solve it optimally. We introduce a notion of *level* of flow paths in our CSP formation to get rid of invalid loops successfully. A CSP solver may not be able to solve the large CSP instances in reasonable time. Thus, in addition to the *CSP-based optimal* method (referred as an exact method), we also propose an efficient heuristic named as *Determining Flows from the Last (DFL)* method. *DFL* is based on an idea that it is better to determine flows from the last, *i.e.*, the order of determining flows is opposite to that of *FLAP* [4]. Simulation results confirm that the *CSP-based optimal* method (the exact method) can be applied to most of the bio-protocols, and the heuristic *DFL* outperforms the state-of-the-art technique *FLAP* [4].

The remainder of the paper is organized as follows. In Sec. II the motivation and the problem formulation are discussed. Then we explain the two proposed methods in Sec. III. The simulation results are presented in Sec. IV and finally, Sec. V concludes the paper.

II. Motivation and Problem Formulation

In this section, we discuss about the motivation and the formulation of the fluid loading problem to be solved in order to execute any bio-protocol on a PMD chip.

A. Motivation

The low-level synthesis of a scheduled sequencing graph for a bio-protocol includes the placement of corresponding modules (mixer, storage, etc.) and the routing of fluids. Su *et al.* [7] presented a collision-free routing algorithm for PMD chips, where parallel routing paths are considered those are assigned to transport fluid sections of different volumes from multiple input ports to corresponding output ports on the periphery of the PMD chip (Fig. 1(a)). Here, collisions are possible only between the routing paths those are overlapped in time. According to the routing paths as shown in Fig. 1(a), the circular shaded region is the collision point between those two routes. In contrast to fluid routing, the fluid loading problem finds the routing paths of the fluids, which must pass through a set of predefined cells in a PMD chip with one input and one output ports. In Fig. 1(b) it is assumed that two reactant fluids (reagents) "B" and "R" are to be loaded in the predefined cells (marked with dotted polygons) of a PMD chip containing only one input and one output ports. After the fluid "B" is already loaded with the first flow (F_1), the shaded cell containing fluid "B" acts as an obstacle for the second flow (F_2) for the loading of fluid "R".

Consider that the sequencing graph of a bio-protocol [3] (as shown in Fig. 2(a)) and the corresponding module placement information (as shown in Fig. 2(b)) for a PMD chip of fixed dimension of $n \times m$ (here $n = 6$ and $m = 5$ in Fig. 2(b)) are given. The desired placement of

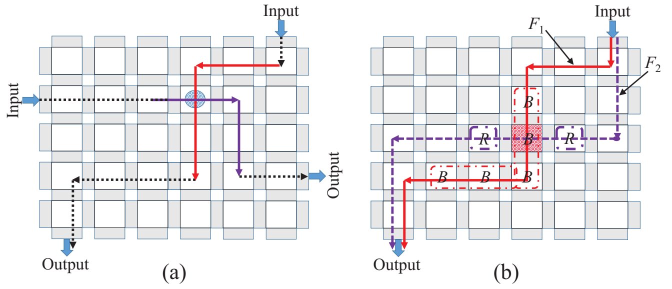

Fig. 1: (a) Simultaneous routing of two different fluid sections. (b) Serial loading of two reagents (labeled as "B" and "R") in a PMD chip of 6×5 dimension.

978-1-7281-4123-7/20/$31.00 © 2020 IEEE

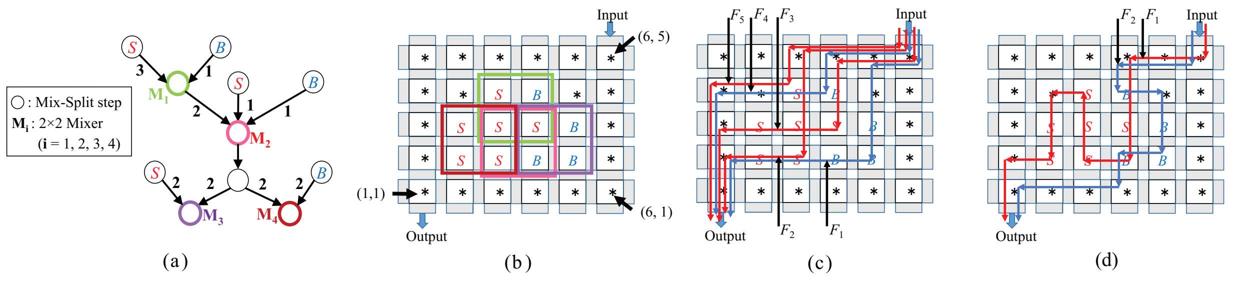

Fig. 2: (a) A sequencing graph of an example bio-protocol considering sample fluid "S" and buffer fluid "B" as input fluids. (b) Desired placement of fluids into the cells of a 6×5 PMD chip for all the mixing steps (same colors have been used to indicate the one-to-one correspondences). For the same fluid loading problem, (c) a solution (five flows) determined by *FLAP* [4], and (d) another solution (two flows) obtained by the *CSP-based optimal* method.

modules requires that T types of different reactant fluids (reagents), denoted by $t_1, t_2, \cdots, t_T$, are to be loaded in different cells on the PMD chip. In Fig. 2, $t_1 = B$ and $t_2 = S$, where "S" stands for sample and "B" stands for buffer. We can put any reagent (or no reagent) in the cells labeled with "$*$" as shown in Fig. 2(b). A cell with a label other than "$*$" (*e.g.,* "S", "B", etc.) means that we need to put that reagent indicated by the label into that PMD cell.

Note that for a PMD chip of $n \times m$ dimensions as shown in Fig. 2, the bottom-left cell is $Cell(1,1)$ and the top-right cell is $Cell(n,m)$. In the following sections, $Cell(i,j)$ is used to denote a cell located at the position (i,j). Without loss of generality, we assume $Cell(n,m)$ is connected to the dispensing (input) port, and $Cell(1,1)$ is connected to the output port of a PMD chip. So, a *flow of any fluid* from $Cell(n,m)$ to $Cell(1,1)$ through any cell can be created by using the programability of a PMD chip. For example, a red path, *e.g.,* F_3 (blue path, *e.g.,* F_1) in Fig. 2(c) is a flow of fluid "S" ("B") from $Cell(6,5)$ (the input port) to $Cell(1,1)$ (the output port).

If some PMD cells on a flow path are occupied by some previously loaded reagents, then the current flow path may result into an unintended mixing of different reagents. Thus, in the real-life implementation of a bio-protocol on a PMD chip, we need a *wash* flow (the flow of a wash fluid) to wash such previously loaded reagents in some PMD cells before loading any new reagent. As shown in Fig. 2(d), after the first flow F_1 is completed, $Cell(4,2)$ and $Cell(4,4)$ loaded with reagent "S" are needed to be washed before the loading of reagent "B" (by flow F_2) starts. After the flow F_1 is completed, the *wash* flow path would be the same as that of F_2. As we will always have to perform this *wash* process to avoid unintended mixing of different reagents during loading, for simplicity, we do not consider the number of wash flows in our problem formulation. Instead, we simply consider that the current flow of a reagent overrides the other reagents previously loaded by any flow.

B. Problem Formulation

The *fluid loading problem* for a PMD chip can be formally defined as follows.

Inputs: An $n \times m$ array of cells of a PMD chip and a desired placement of on-chip mixing modules, *i.e.*, T types of different reactant fluids (reagents), denoted by $t_1, t_2, \cdots, t_T$, are to be loaded in different cells on the PMD chip.

Output: A sequence of fluid-flows $F_1, F_2, \cdots, F_K$ that loads all the reagents in the PMD cells as desired by the module placement.

Objective: To find the solution with the minimum number of flows (*i.e.*, K in this case) that complete the loading of fluids in the PMD.

III. Fluid Loading in PMD Chips

In this section, we present an exact method called as *CSP-based optimal* method and a heuristic method called as *Determining Flows from the Last*, *i.e.*, *DFL* method, for solving the fluid loading problem on a PMD chip. We discuss these two proposed methods in the subsequent subsections. Furthermore, a combination of both of these methods is also discussed here, which incorporates the benefits of both the exact and heuristic methods.

A. CSP-based Optimal Method: Exact Method

In this section, we explain the proposed exact method to find the minimum number of fluid-flows to load the reagents. To find the minimum number, our strategy is as follows: we first formulate the condition that K flows can load the reagents successfully as a constraint satisfaction problem (CSP) [6], and then we use a CSP solver to check whether or not the condition is satisfied. We can determine the minimum number of flows by solving the CSPs with different values of K repeatedly.

Here, we explain how to formulate the CSP using the following variables/notations:

$S_{(i,j)}$	denotes a reagent type number to be loaded at $Cell(i,j)$, *e.g.*, if we want to load t_2 at $Cell(i,j)$, then $S_{(i,j)} = 2$.
$X_{k,l}$	is a 0/1 variable, which becomes 1 if the k^{th} flow is for the reagent type t_l, otherwise, it becomes 0. By the definition, we need the condition such that for all l, only one of the $X_{k,l}$s should become 1.
$Z_{(i,j),k}$	is a 0/1 variable, which becomes 1 if the k^{th} flow pass through $Cell(i,j)$, otherwise, it becomes 0.

Successful Fluid Loading Condition:

By using the above variables, we can express the conditions to load the reagents successfully by K flows as Equation (1).

$$
\begin{aligned}
&S_{(i,j)} = l \Rightarrow \\
&(Z_{(i,j),K} = 1 \land X_{K,l} = 1) \lor \\
&(Z_{(i,j),K} = 0 \land Z_{(i,j),K-1} = 1 \land X_{K-1,l} = 1) \lor \\
&(Z_{(i,j),K} = 0 \land Z_{(i,j),K-1} = 0 \\
&\qquad \land Z_{(i,j),K-2} = 1 \land X_{K-2,l} = 1) \lor \\
&\quad \vdots \\
&(Z_{(i,j),K} = 0 \land Z_{(i,j),K-1} = 0 \cdots \\
&\qquad Z_{(i,j),2} = 0 \land Z_{(i,j),1} = 1 \land X_{1,l} = 1)
\end{aligned}
\tag{1}
$$

Here in Equation (1), $S_{(i,j)} = l$ means that we need to load a reagent of type t_l at $Cell(i,j)$. If the k^{th} flow (*i.e.*, the last flow) goes to $Cell(i,j)$, then the flow should be of reagent with type t_l. This condition corresponds to the second row in the above equation, *i.e.*, $(Z_{(i,j),k} = 1 \land X_{k,l} = 1)$. If the k^{th} flow does not go to $Cell(i,j)$ and the $(k-1)$-th flow goes to $Cell(i,j)$, then the $(k-1)$-th flow should be of reagent with type t_l. This condition corresponds the third row in the above equation, *i.e.*, $(Z_{(i,j),k} = 0 \land Z_{(i,j),k-1} = 1 \land X_{k-1,l} = 1)$. We consider the similar conditions until the situation where the first flow load the reagent of type t_l at $Cell(i,j)$, which corresponds to the last row in the above Equation (1).

Next, we need to consider the condition of each flow to be valid. For that purpose, we use the following variables/notations:

$U_{(i,j),k}$	is a 0/1 variable, which becomes 1 if the k^{th} flow goes into $Cell(i,j)$ from the upper side of the cell, otherwise it becomes 0.
$B_{(i,j),k}$	is a 0/1 variable, which becomes 1 if the k^{th} flow goes into $Cell(i,j)$ from the lower side of the cell, otherwise it becomes 0.

$L_{(i,j),k}$ is a 0/1 variable, which becomes 1 if the k^{th} flow goes into $Cell(i,j)$ from the left side of the cell, otherwise it becomes 0.

$R_{(i,j),k}$ is a 0/1 variable, which becomes 1 if the k^{th} flow goes into $Cell(i,j)$ from the right side of the cell, otherwise it becomes 0.

By using these variables, we can express the conditions of each fluid-flow path to be valid as the following Equations (2) to (6).

Valid Flow Path Condition:

When $Z_{(i,j),k} = 1$, the k^{th} flow path should go into $Cell(i,j)$. For this condition, we need the following.

$$Z_{(i,j),k} = 1 \Rightarrow U_{(i,j)k} + L_{(i,j)k} + R_{(i,j)k} + B_{(i,j)k} = 1 \quad (2)$$

If the k^{th} flow goes into $Cell(i,j)$, then the flow should continue to go into one of its three adjacent cells, which are not visited by the flow. For this condition, we need:

$$U_{(i,j),k} = 1 \Rightarrow (L_{(i+1,j),k} + R_{(i-1,j),k} + U_{(i,j-1),k}) = 1 \quad (3)$$

$$L_{(i,j),k} = 1 \Rightarrow (L_{(i+1,j),k} + U_{(i,j-1),k} + B_{(i,j+1),k}) = 1 \quad (4)$$

$$R_{(i,j),k} = 1 \Rightarrow (R_{(i-1,j),k} + U_{(i,j-1),k} + B_{(i,j+1),k}) = 1 \quad (5)$$

$$B_{(i,j),k} = 1 \Rightarrow (R_{(i-1,j),k} + B_{(i,j+1),k} + L_{(i+1,j),k}) = 1 \quad (6)$$

At the surroundings of a chip, there may be no adjacent cell. In that case, we remove the corresponding variable in the above equations accordingly, if there is no adjacent cell.

Note that, without loss of generality, our problem formulation assumes that $Cell(n,m)$ is connected to the input port, and $Cell(1,1)$ is connected to the output port of the PMD chip under consideration. Thus, we need to add the following conditions for the k^{th} fluid-flow: $U_{(n,m),k} = 1$, and only one of $U_{(1,1),k}$ and $R_{(1,1),k}$ should be 1.

No Loop Condition:

It is easy to see that the above-mentioned conditions guarantee the following.

- Each flow path starts from the input port, and arrives at the output port.
- Each flow path does not split at any cell.
- Each reagent is loaded to a specified cell by at least one flow, and after the last flow to load that reagent, another flow does not go to that cell.

Thus, it seems that we have already listed up the necessary conditions in the above. Unfortunately, it is not true; there is an invalid situation that satisfies all the conditions mentioned above. Consider an example shown in Fig. 3, where the k^{th} flow (shown by the red arrows) is loading reagent "S". It is easy to see that all the above conditions are satisfied by the assignment of variables corresponding to the situation in Fig. 3 because there is no split in the flow path, and there is a valid flow path from the input port to the output port. Moreover, the flow can go to all the cells where "S" should be loaded. Thus, we would wrongly conclude that all "S" can be loaded by only one flow by using our CSP formulation as above. However, the flow path (denoted by the red arrows in Fig. 3) obviously is invalid because it contains a flow in loop, which is not connected to the input/output port (*e.g.,* the loop consisting of $Cell(2,4), Cell(2,3), Cell(3,3), Cell(3,4)$ in Fig. 3). Hence,

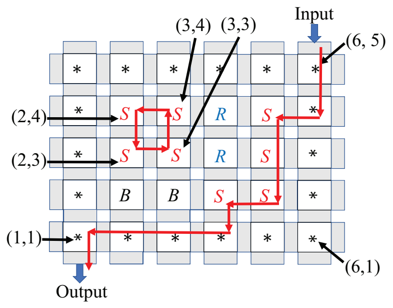

Fig. 3: Example of an invalid loop while solving the fluid loading problem by the *CSP-based optimal* method.

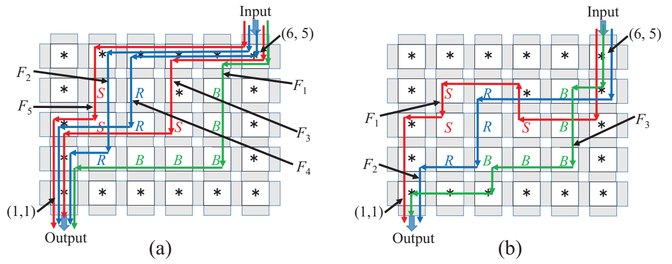

Fig. 4: The same fluid loading problem on a 6×5 PMD chip solved by (a) *FLAP* [4] and (b) *CSP-based optimal* (exact) method with total number of flows (K) required to load the reagents as 5 and 3, respectively.

we cannot exclude such a loop by using only the aforementioned formulation.

To exclude a flow in loop (loop flow), our idea is to calculate the *level* of each cell for each flow path and we can have a condition to exclude a loop by using the values of level. First, we prepare an integer variable $level_{(i,j),k}$ to denote the level of $Cell(i,j)$ for the k^{th} flow. Then, we add the following condition to calculate the level recursively.

- $level_{(l,m),k} = level_{(i,j),k} + 1$, iff the k^{th} flow goes to $Cell(l,m)$ right after $Cell(i,j)$. (This condition implicitly assumes that $Cell(l,m)$ and $Cell(i,j)$ are adjacent with each other, and we need to have this condition for each pair of two adjacent cells.)

We also have the condition: $level_{(m,n),k} = 0$, which defines the level of the starting cell as 0. Then, the value of level is increased by 1, when the flow enters into the next cell following the above condition. Assume that, a loop flow of length L starting at $Cell(i,j)$. Following this last condition, the level is increased by 1 along the flow path. The increment of the level is repeated L times before the path comes back to $Cell(i,j)$ by the above condition. Then eventually we have $level_{(i,j),k} = level_{(i,j),k} + L$, which cannot be satisfied by any variable assignment. Thus, we can exclude a loop flow with the help of this new condition.

We can find an assignment of variables that satisfies all the conditions using a CSP solver, if there is such an assignment. We can easily determine the k^{th} flow by the assigned values of $U_{(i,j),k}$, $B_{(i,j),k}$, $L_{(i,j),k}$, $R_{(i,j),k}$ those are determined by the CSP solver. Therefore, we can also find the smallest number of flows by invoking the CSP solver many times with different values of K.

For a fluid loading problem with three reagents "S", "R" and "B", *FLAP* requires $K = 5$ flows as shown in Fig. 4(a), whereas the *CSP-based optimal* (exact) method determines a sequence of fluid-flows with the total number of flows as $K = 3$ (Fig. 4(b)).

B. *An Efficient Heuristic Method: DFL Method*

Although our exact method should work for most of the cases, it may not find a solution for a large size problem instance. In such a case, we need to have a heuristic method as *FLAP* [4]. However, we observe that the number of flows can be further reduced; if we carefully consider the effect of a flow, *i.e.*, a flow *overrides* the reagents those are already loaded by previous flows. In order to utilize this idea, we propose a more efficient heuristic method that determines flows from the last, *i.e.*, the order of determining flows is opposite to *FLAP*. We name this heuristic method as *Determining Flows from the Last* (*DFL*) method.

Here, we explain the idea behind the proposed heuristic *DFL* method. Consider the flow F_1^{last} in Fig. 5(a) is the *last* flow, which is the first flow in *FLAP*. The flow F_1^{last} loads the reagent of type "B" on its path. This means that it overrides the reagents in the cells on its path indicated by green lines); *i.e.*, any flow before F_1^{last} is allowed to load a reagent of any type into a cell on the path of F_1^{last}. Therefore, for our example, we can consider the 'L-shaped' area (encircled by a green dotted box) as "$*$" as shown in Fig. 5(b) for the flows before F_1^{last}. By this consideration, we have more freedom

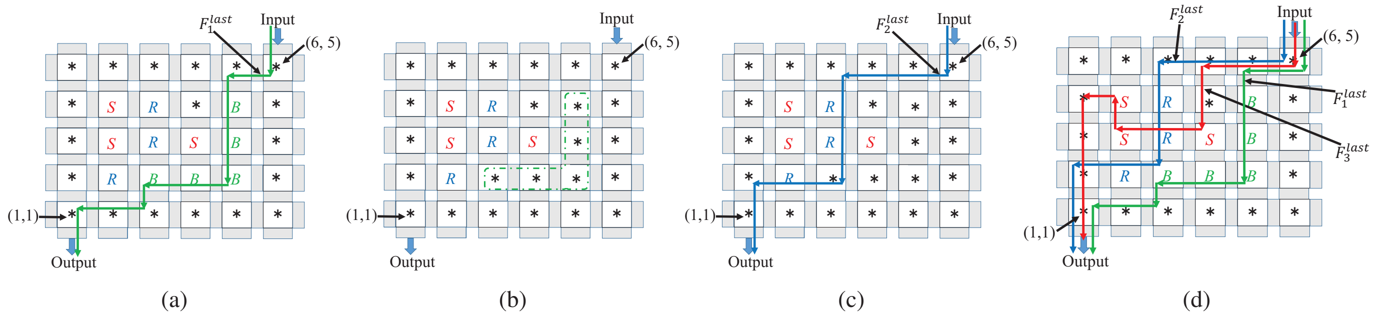

Fig. 5: Solution to an example fluid loading problem obtained by *DFL* method. (a) The last flow, (b) situation before the last flow, (c) second flow to the last flow, and (d) all flows.

to find a "good" flow before F_1^{last}. Based on this consideration, our overall strategy is as follows:

- We determine the flows in *reverse* order unlike the normal methods.
- We consider the cells on the already determined flows as "$*$", and find a "good" flow, which loads the reagents before the already determined flows.

In this paper, we consider a "good" flow should contain as many cells of the reagent type as possible, and also a shorter path would be better. Hereafter, we use the following notation:

- $Weight_{t_i}(F_j)$ is the number of cells of reagent type t_i on the path of a flow F_j.

Algorithm 1 provides the pseudo-code of the ***DFL* Method**. First, we initialize $FlowList$ at line 1 as empty, and then we try to find the good flows one-by-one in the **while** loop between lines 2 to 14 until there is no *undetermined cell*, which means that we have found all the necessary flows for the given fluid loading problem. As we mentioned, unlike normal methods, we try to find a flow from the last. So we add a flow to the beginning of $FlowList$ at line 12. After we find a flow path, we consider all the cells on that path to have "$*$" labels, and continue to find flow paths until there is no undermined cell. At line 6, the "best" flow is found to load the reagent of type t_i by **BestFlow**(t_i), which will be explained later. Between lines 4 and 11, we try to find the "best" flow for each reagent type and then we choose the best one among all types as $BestFlowFound$ by replacing it at line 8, when we find a better one.

Algorithm 2 describes the method **BestFlow**(t_i), which is a heuristic to find a possibly "best" flow to load the reagent of type t_i. Here, we consider a flow F is good, if $Weight_{t_i}(F)$ is large. Here, we explain how we find a flow for reagent "R", when the situation of cells is as shown in Fig. 5(c). The method **BestFlow**(R) finds the flow F_2^{last} indicated by blue arrows in Fig. 5(c), which is the second last flow in the given fluid loading problem. For the same problem with three reagents "S", "R" and "B" as shown in Fig. 4 for *FLAP* and the exact method, here the *DFL* method determines a sequence of flows as depicted in Fig. 5(d) with the total number of flows as $K = 3$.

First, we find $Cell_{start}$, which is a cell with label t_i (for reagent type t_i) and is reached first from the output, *i.e.*, $Cell(1,1)$ at line 1. Note that for the problem instance shown in Fig. 5(c), in order to find a flow of reagent type "R", we consider that a flow can go through only cells with labels "R" or "$*$". It is easy to find such a cell by just extending paths from $Cell(1,1)$ to reach at a cell with label "R". In this example, $Cell_{start}$ is $Cell(2,2)$ as shown in Fig. 6(a) indicated by a circle around reagent "R". The next task is to find a flow path containing as many "R" as possible from $Cell_{start}$ to the input port $Cell(6,5)$. The path from $Cell(1,1)$ to $Cell_{start}$ is indicated by red arrows in Fig. 6(a), and thus we cannot use this path for the path to the input port. Hence, we put "P" labels in the cells as shown in this figure, which means these cells are also obstacles for a path from $Cell_{start}$ to the input port.

For the remaining task, our idea is as follows. We consider every cell has four incoming and four outgoing edges, which are connected to its neighboring cells as shown in Fig. 6(a). Each edge has a value

Algorithm 1: *DFL* Method: Finding a list of flows required for loading the desired placement of reagents.

```
1:  $FlowList = empty$;
2:  while there exists an undetermined cell do
3:      $BestWeight = 0$;  /* Number of reagents in the best flow
        found so far */
4:      for all $t_i \in \{t_1, \cdots, t_T\}$ do
5:          $BestFlowFound = empty$;
6:          $FlowFound$ = BestFlow($t_i$);
7:          if $Weight_{t_i}(FlowFound) > BestWeight$ then
8:              $BestFlowFound = FlowFound$;
9:              $BestWeight = Weight_{t_i}(FindFlow)$;
10:         end if
11:     end for
12:     Add $BestFlowFound$ to the beginning of $FlowList$;
13:     Update all the cells with label "$*$" on the path of
        $BestFlowFound$;
14: end while
15: return  $FlowList$;
```

Algorithm 2: BestFlow(t_i): Finding a best flow to load reagents of type t_i.

```
1:  Find a cell $Cell_{start}$ with label $t_i$ such that there is a path
    consisting of only "$*$"s from $Cell(1,1)$ to $Cell_{start}$, and
    $Cell_{start}$ is the nearest to $Cell(1,1)$;
2:  if  there is no such $Cell_{start}$ then
3:      return  Null;
4:  end if
5:  $Flag$ = True;
6:  while $Flag$ == True do
7:      UpdateCells($t_i$);
8:      if there is any change in UpdateCells then
9:          $Flag$ = True;
10:     else
11:         $Flag$ = False;
12:     end if
13: end while
14: if the outgoing edge from input port cell has a positive value
    then
15:     $BestFlow$ is set as a path related to the outgoing edge from
        the input port cell;
16:     return  $BestFlow$;
17: else
18:     return  Null;
19: end if
```

that means the number of "R"s contained in the path from $Cell_{start}$. At first, every edge has value 0 except for the outgoing edges from $Cell_{start}$ as shown in Fig. 6(a). The outgoing edges from $Cell_{start}$ has value 1, and this value is propagated towards the input port. While propagating the values, if we enter into a cell with label t_i

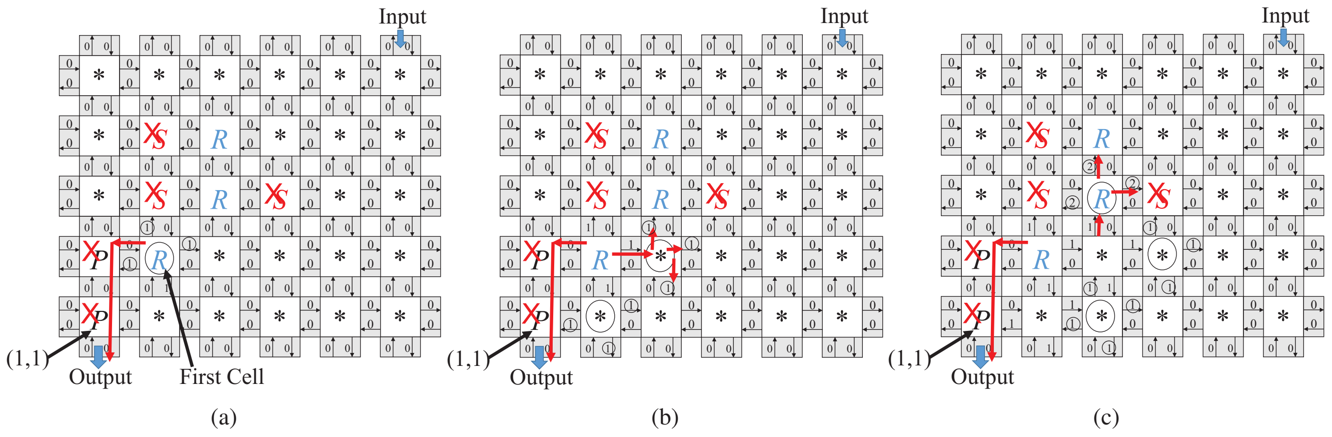

Fig. 6: Use of the function UpdateCells() in *DFL* method for problem instance shown in Fig. 5(c). (a) Initial situation, (b) the first call of UpdateCells() and (c) the second call of UpdateCells().

("R" in this example), the value is increased by 1 because the path now finds a cell with "R". This propagation should be stopped by the cells with labels other than t_is or "$*$" as indicated by red "X"s in Fig. 6(a). If we try all the possible propagation processes, we will find the path containing the maximum number of cells with reagent t_i.

However, this should be time consuming, so our idea is to omit the propagation of the values *only* when they are increased. The above propagation is done by UpdateCells(t_i) at line 7. The function UpdateCells(t_i) simulates each flow to go through only one cell. As we update values only when they are increased, we check only the cells such that the values of their incoming edges are updated (increased) at the previous call of UpdateCells. In our example shown in Fig. 6(b), at the first call of UpdateCells we check only $Cell(2,1)$ and $Cell(3,2)$, which are adjacent to $Cell_{start}$ and indicated by circles around the "$*$"s. This is because only their incoming edges have value 1 at the initial situation as indicated in Fig. 6(b). More formally, UpdateCells updates the values of the outgoing edges by the following way:

- If the value of one of the incoming edges is updated (increased), we update the values of the outgoing edges in three other directions. The updated value is the same as the incoming edge for a cell with label "$*$". The value is increased by 1 for a cell with label t_i.

In the example of Fig. 6, UpdateCells updates the values of the outgoing edges of $Cell(2,1)$ and $Cell(3,2)$ to 1 at the first call. The updated values are circled in Fig. 6(b). At the next call of UpdateCells, we update the values of the outgoing edges of the three cells, which are indicated by circles in Fig. 6(c). The updated values on the outgoing edges are surrounded by circles. The values of the outgoing edges of $Cell(3,3)$ are updated as 2 because the flow, which stars from $Cell(1,1)$ and goes throughout this cell, contains two "R"s.

We should be careful not to make a flow *loop* during the above process. To do so we attach the path information to each value on an edge. For example, consider the outgoing edge of $Cell(3,1)$ to $Cell(3,2)$ after the second call of UpdateCells as shown in Fig. 6(c). The value of this edge has just updated to 1, which is indicated by a circle in the figure. This updated value 1 corresponds to a path: $\{Cell(2,2), Cell(2,1), Cell(3,1)\}$, which has one "$R$". We keep this information with the value of the edge.

At the third call of UpdateCells after Fig. 6(c), we consider to update the outgoing edges of $Cell(3,2)$ because it has the updated incoming edge from $Cell(3,1)$. However, we do not update the values of the outgoing edges to $Cell(4,2)$ and $Cell(3,3)$ because they are already 1. Moreover, we do not update the value on the edge to $Cell(2,2)$ from the current vale 0 to 1. This is because if we do so, we will make a loop flow. It is easy to know this by checking whether the path information of the incoming edge, *i.e.*, $\{Cell(2,2), Cell(2,1), Cell(3,1)\}$ has $Cell(2,2)$ or not.

Finally, we do not update the values when we make a loop flow, but we update the values only when they are increased. So, it is easy to see that the number of calls of UpdateCells should not exceed the minimum length of the paths that contains the maximum number of labels t_is. Thus the algorithm can be practically used. Note that the algorithm is not theoretically optimal because we check the change of incoming edges only when their values are increased.

If there is no more change by UpdateCells, then it goes to line 14 in **Algorithm 2**, and get the best flow as *BestFlow* to return it. If there is no possible flow path from $Cell_{start}$ to the input port (because of obstacles), we cannot find a flow. Hence, when we go out the **while** loop between line numbers 6 to 13, the outgoing edge of the input port cell will have a value as 0 and an empty list as **Null** is returned at line 18. In some cases, we may not find a path for a particular reagent. For example, if all "S"s are enclosed by other reagent(s), then we cannot have any flow path for reagent "S", whose weight is greater than zero. Here, first we load the other reagent(s), by which these cells become "$*$", and then we can have a flow path to load reagent "S".

C. *Combination of the Exact Method and Heuristic Method*

If we can find the smallest number of flows to load the reagents by the exact method, then obviously there is no problem. However, we consider that the exact method cannot treat a large size fluid loading problem because the method needs to solve a CSP. Indeed, the number of variables in this CSP formulation is obviously $O(n \cdot m \cdot K)$, where K is the number of flows and the chip size is $n \times m$. Again, the formula size (*i.e.*, the number of variables in the formula) of Equation (1) is $O(K^2)$. Thus, the CSP formulation becomes larger, when the chip size and/or the value of K becomes larger.

Instead, the heuristic method is not time consuming, which will also be confirmed by the simulation results in Sec. IV. The reason is that the number of repetitions of calls of the function UpdateCells in *DFL* method is bounded by the length of the longest path, which is linear to the chip size.

Here, we propose one possible way to utilize the heuristic method along with the exact method.

A Practical Method Utilizing both Heuristic and Exact Methods:

Step 1 Invoke a heuristic method (note that any heuristic would work) to solve the problem. Let the number of flows found by the heuristic be K_h.

Step 2 Invoke the exact method with the number of flows to be $(K_h - 1)$. If a CSP solver confirms that the CSP is not satisfied in reasonable time, we consider that the solution by the heuristic method is optimal, and terminate this algorithm. Otherwise, go to Step 3.

Step 3 Invoke the exact method to find the number $K_{optimal}$ such that the CSP with $K_{optimal}$ is satisfied, but the CSP with $(K_{optimal} - 1)$ is not satisfied for the current problem instance. We can find such a number by a binary-search

TABLE I: Simulation results of 10 test cases for the exact method and the heuristic *DFL* over *FLAP* [4].

Sl. No.	**Test Case**	**Chip Size**	**Exact**		**Heuristic**		***FLAP***	
			K	**Time (in *seconds*)**	K	**Time (in *seconds*)**	K	**Time (in *seconds*)**
1	*cell-culture* [8]	9×9	3	571.643	5	0.014	12	0.002
2	*pcr-mix* [3]	6×6	8	7.528	8	0.008	8	0.002
3	*protein* [3]	7×8	3	15.061	3	0.008	9	0.002
4	*invitro1* [3]	8×10	7	–	7	0.024	15	0.002
5	*invitro2* [3]	10×10	8	–	8	0.052	20	0.033
6	*rsm* [9]	4×9	3	3.301	3	0.006	7	0.002
7	*mtcs* [10]	6×7	3	2.746	4	0.006	9	0.002
8	*flospa-em* [11]	5×5	5	1.26	5	0.005	8	0.002
9	*synthetic1*	5×5	4	1.782	5	0.005	8	0.001
10	*synthetic2*	5×6	2	2.177	3	0.004	5	0.001

strategy. If a CSP solver can finish in a reasonable time, we consider $K_{optimal}$ is the optimal number of flows and terminate this algorithm. Otherwise, go to Step 4.

Step 4 Find one best flow, which is the last flow as our *DFL* method provides, for the current problem instance by a heuristic method. Let the found flow to be the last flow, and we consider that reagent type on the flow to be "$*$" in the current problem instance to make a smaller problem. Then go to Step 3 with the smaller size problem instance to be solved at Step 3.

The idea is that if a given problem is larger for the exact method, we determine the last flow by a heuristic method, and then the remaining problem becomes smaller, with which we can try the exact method. Thus, we continue to make the problem smaller until the exact method can be applied to get the solution in reasonable time.

IV. Simulation Results

We evaluated the proposed heuristic *DFL* and the exact method (*CSP*-based) with the *FLAP* [4]. For performance comparison, we considered total ten (10) test cases (out of which eight are for the sequencing graphs corresponding to the bio-protocols taken from the literature and two are synthetically generated by us) namely *cell-culture* [8], *pcr-mix* [3], *protein* [3], *invitro1* [3], *invitro2* [3], *rsm* (mixing tree for target ratio 7 : 14 : 11 obtained by *RSM*) [9], *mtcs* (mixing tree for target ratio 7 : 14 : 11 obtained by *MTCS*) [10], *flospa-em* (mixing tree for target ratio 22 : 14 : 14 : 14 obtained by *FloSPA-EM*) [11], *synthetic1* and *synthetic2*.

Table I presents the number of flows (K) required to solve the fluid loading problems in the PMD chips corresponding to these ten bio-protocols obtained by the exact method, the heuristic method and *FLAP* [4], in the fourth, the sixth and the eighth columns, respectively. The fifth, seventh and ninth columns report the CPU time (in seconds) of the exact method, the heuristic method *DFL* and *FLAP*, respectively, while executing the programs with Ubuntu 18.04 OS in a computer with Intel(R) Core(TM) i7-6700K CPU @ 4.0GHz and 8GB memory. We used Sugar [12] to solve the CSP in the exact method.

The proposed heuristic *DFL* can find better solutions than *FLAP*, and indeed it can find almost optimal solutions for most of the test cases. As expected, in case of the two test cases namely *invitro1* and *invitro2*, we found that the exact method could not determine the flows even within one hour, whereas the same number of flows (7 and 8 flows, respectively) are determined by the heuristic method *DFL* within a second. Hence, we used "–" in Table I to indicate that the exact method needs much more CPU time. Thus, for these two bio-protocols, we can confirm that *DFL* can find the optimal solutions same as obtained by the exact method. As we discussed at the end of the last section, in case of large size test cases, we can use the exact method in combination with *DFL*.

V. Conclusions and Future Work

This paper proposes an exact method and a near-optimal heuristic method *DFL* for automation of fluid loading into the cells on a PMD chip. Simulation results confirm that the exact method can be useful for most of the bio-protocols, and the proposed heuristic *DFL* can provide the near-optimal solutions as obtained by the exact method. Both the methods outperform the state-of-the-art technique *FLAP*. As a future work, one can consider the existence of multiple input and output ports, and solve the simultaneous fluid loading problems on a PMD chip.

Acknowledgment

This work by D. Kundu and S. Roy was supported partially by the early career research award (grant #: ECR/2016/001921) sponsored by SERB, Govt. of India and the Indo-Japan joint research project (grant #: DST/INT/JSPS/P-256/2017) sponsored by DST, Govt. of India. This work by S. Yamashita was supported partially by the Bilateral Joint Research Project sponsored by JSPS, Govt. of Japan.

References

[1] N. Amin, W. Thies, and S. Amarasinghe. Computer-Aided Design for Microfluidic Chips based on Multilayer Soft Lithography. In *Proc. of the ICCD*, pages 2–9, 2009.

[2] L. M. Fidalgo and S. J. Maerkl. A Software-Programmable Microfluidic Device for Automated Biology. *Lab Chip*, 11:1612–1619, 2011.

[3] DMFBSSS: Digital Microfluidic Biochip Static Synthesis Simulator, Microfluidics at UCR, May 2017. http://microfluidics.cs.ucr.edu/dmfb_static_simulator/overview.html.

[4] A. Gupta, J.-D. Huang, S. Yamashita, and S. Roy. Design Automation for Dilution of a Fluid Using Programmable Microfluidic Device-Based Biochips. *ACM TODAES*, 24(2):21:1–21:24, 2019.

[5] S. Roy, B. Bhattacharya, and K. Chakrabarty Optimization of Dilution and Mixing of Biochemical Samples using Digital Microfluidic Biochips. *IEEE TCAD*, 29(11):1696–1708, 2010.

[6] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys (CSUR)*, 38(4):12, 2006.

[7] Y. Su and T.-Y. Ho and D. Lee. A Routability-Driven Flow Routing Algorithm for Programmable Microfluidic Devices. In *Proc. of the ASP-DAC*, pages 605–610, 2016.

[8] R. J. Taylor, D. Falconnet, A. Niemistö, S. A. Ramsey, S. Prinz, I. Shmulevich, T. Galitski, and C. L. Hansen. Dynamic Analysis of MAPK Signaling Using a High-Throughput Microfluidic Single-Cell Imaging Platform. *Proc. of the National Academy of Sciences*, 106(10):3758–3763, 2009.

[9] Y.-L. Hsieh, T.-Y. Ho, and K. Chakrabarty. A Reagent-Saving Mixing Algorithm for Preparing Multiple-Target Biochemical Samples Using Digital Microfluidics. *IEEE TCAD*, 31(11):1656–1669, 2012.

[10] Shalu, S. Kumar, A. Singla, S. Roy, K. Chakrabarty, P. P. Chakrabarti, and B. B. Bhattacharya. Demand-Driven Single- and Multitarget Mixture Preparation using Digital Microfluidic Biochips. *ACM TODAES*, 23(4):55:1–55:26, 2018.

[11] S. Bhattacharjee, S. Poddar, S. Roy, J. D. Huang, and B. Bhattacharya. Dilution and Mixing Algorithms for Flow-Based Microfluidic Biochips. *IEEE TCAD*, 36(4):614–627, 2017.

[12] N. Tamura and M. Banbara. Sugar: A CSP to SAT translator based on order encoding. *Proc. of the Second International CSP Solver Competition*, pages 65–69, 2008.