Module-Based Synthesis of Digital Microfluidic Biochips with Droplet-Aware Operation Execution

ELENA MAFTEI, PAUL POP, and JAN MADSEN, Technical University of Denmark

Microfluidic biochips represent an alternative to conventional biochemical analyzers. A digital biochip manipulates liquids not as continuous flow, but as discrete droplets on a two-dimensional array of electrodes. Several electrodes are dynamically grouped to form a virtual device, on which operations are executed by moving the droplets. So far, researchers have ignored the locations of droplets inside devices, considering that all the electrodes forming the device are occupied throughout the operation execution. In this article, we consider a droplet-aware execution of microfluidic operations, which means that we know the exact position of droplets inside the modules at each time-step. We propose a Tabu Search-based metaheuristic for the synthesis of digital biochips with droplet-aware operation execution. Experimental results show that our approach can significantly reduce the application completion time, allowing us to use smaller area biochips and thus reduce costs.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Biochips, microfluidics, synthesis

ACM Reference Format:
Maftei, E., Pop, P., and Madsen, J. 2013. Module-Based Synthesis of Digital Microfluidic Biochips with Droplet-Aware Operation Execution. *ACM J. Emerg. Technol. Comput. Syst.* 9, 1, Article 2 (February 2013), 21 pages.
DOI: http://dx.doi.org/10.1145/2422094.2422096

1. INTRODUCTION

According to Moore's law [Moore 1965] the number of transistors on an integrated circuit doubles approximately every two years. "More than Moore" explores new applications in which such systems can be used, focusing on function diversification rather than increasing density. An emerging field related to embedded systems is the design of efficient, low-cost devices for the biomedical area, which has been highlighted by the International Technology Roadmap for Semiconductors 2007 [ITRS07] as an important system driver for the near-future [Chakrabarty et al. 2010].

In recent years, microfluidic biochips (also called labs-on-chips) have emerged as a miniaturized alternative to conventional laboratories. On such devices, biochemical applications can be performed using small amounts of fluids, in the range of micro- or nanolitres. In addition to lower reagent costs compared to conventional laboratories, biochips can be fully automated and provide higher sensitivity.

This article is an extended and revised version of the paper presented in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'09)*, 195–203.
Authors' address: E. Maftei, P. Pop, and J. Madsen, Technical University of Denmark, 2800, Kgs. Lyngby, Denmark; email: paul.pop@imm.dtu.dk.

DOI: http://dx.doi.org/10.1145/2422094.2422096

Microfluidic biochips can be used in a wide range of fields such as clinical diagnosis, DNA analysis, protein assays and immuno-assays [Chakrabarty 2010]. Considering the potential of such devices for the biotechnology industry, the complexity of biochips is expected to increase, with thousands of operations executed concurrently. In order to support the increase in complexity of these devices and therefore their market growth, computer aided design (CAD) tools are required, which can offer the same level of support as the one taken for granted currently in the semiconductor industry. Initially, designers have used a bottom up approach for the design of biochips, combining fluidic components to create specific-application devices [Fair 2007]. However, this bottom-up approach does not scale to the new designs. Consequently, top-down design methods have been proposed in Chakrabarty and Zeng [2007], increasing the level of abstraction in biochip synthesis. Such techniques are necessary in order to improve the design of biochips, and to hide the implementation details of running biochemical assays from the users [Chakrabarty 2010].

This article focuses on the synthesis of digital microfluidic biochips (DMBs). On these devices, fluids are manipulated as discrete droplets on a two-dimensional array of identical cells, without the need of micro-structures, which offers flexibility and reconfigurability.

1.1. Related Work

On a digital biochip, operations such as mixing and dilution are performed on the microfluidic array by routing the corresponding droplets on a series of electrodes. Two approaches have been considered so far for operation execution. The module-based approach considers that a microfluidic operation is performed by routing the corresponding droplet on a group of adjacent electrodes, forming a virtual device. A different approach, called routing-based synthesis, has been proposed by us in Maftei et al. [2010a], where the concept of modules has been eliminated, allowing droplets to move on any route during operation execution.

This article is based on the module-based synthesis approach. Researchers have initially addressed architectural-level and physical-level synthesis of DMBs separately. In one of the first papers on this topic, an ILP and two heuristic techniques (a modified List Scheduling algorithm and a Genetic Algorithm) have been proposed for the architectural-level synthesis of biochips [Su and Chakrabarty 2004]. The methods consider the problem of scheduling under resource constraints, by roughly estimating the placement of devices on the microfluidic array. The results in Su and Chakrabarty [2004] have been improved in Ricketts et al. [2006], by using a hybrid Genetic Algorithm for scheduling operations under resource constraints.

Although it reduces the complexity of the synthesis problem, the separation of architectural and physical-level synthesis has disadvantages, leading in many cases to a longer completion time of the applications on the biochips [Maftei et al. 2008]. Therefore, the next step taken by researchers was considering a unified approach for the architectural-level synthesis and placement for digital microfluidic biochips.

The first unified methodology was proposed in Su and Chakrabarty [2005], by using a combination of Simulated Annealing and Genetic Algorithms. The focus of the developed method has been on deriving an implementation that can tolerate faulty electrodes. Xu and Chakrabarty [2007] have extended the work done by Su and Chakrabarty [2005], by incorporating routing-awareness during the architectural-level synthesis and placement of modules. The results obtained in Su and Chakrabarty [2005] have been improved in Yuh et al. [2007], by using a tree-based topological representation. The floorplanning algorithm has also been extended to take into account the reconfigurability of biochips in case of defective electrodes. In Maftei et al. [2009] we proposed a Tabu Search-based algorithm for the unified synthesis problem and we

have shown in Maftei [2011] that our method obtains better results than the T-tree approach presented in Yuh et al. [2007].

So far, researchers have assumed that during operation execution in module-based synthesis the droplet repeatedly follows the same pattern inside the virtual module, leading to an operation completion time determined through experiments. The actual position of the droplet inside the virtual device has been ignored, by considering that all the electrodes forming the device are occupied throughout the operation execution. In order to avoid the accidental merging of droplets it was considered that a device is surrounded by a 1-cell segregation area, containing cells that cannot be used until the operation performing on the device is completed.

In this article, we consider a *droplet-aware execution of microfluidic operations*, which means that we know the exact position of droplets inside the modules at each time-step, and we can control them to avoid accidental merging, if necessary.

1.2. Contribution

In this article, we propose a synthesis approach based on a Tabu Search metaheuristic, which, starting from a biochemical application modeled as a sequencing graph and a given biochip array, determines the allocation, resource binding, and scheduling of the operations in the application at the same time as module placement.

Our scheduling and placement steps consider the positions of droplets inside virtual devices during their execution. This allows us to better utilize the chip area, since no segregation cells are needed to separate the modules, and improve the routing step, since the routes can now cross over modules, if needed. Another advantage of droplet-aware operation execution, is that it allows the partial overlapping of modules, which can increase parallelism. However, in this article we do not consider module overlapping, which is left for future work. We show that our droplet-aware operation execution approach can significantly reduce the application completion time compared to the black-box approach.

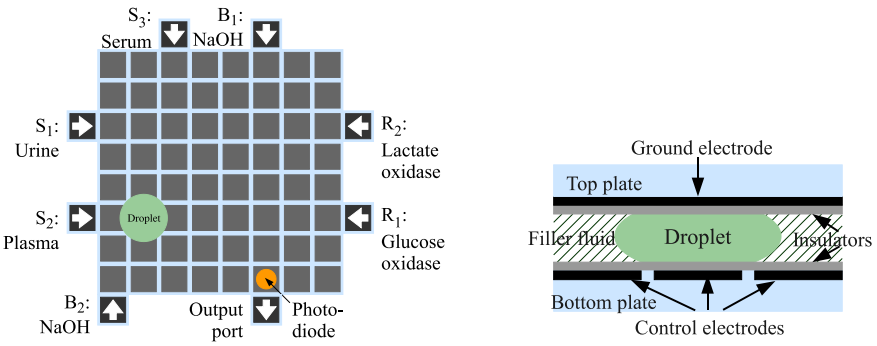
The article is organized in six sections. Section 2.1 presents the architecture of a digital microfluidic biochip. We introduce the abstract model used to capture a biochemical application in Section 2.2. We formulate the problem in Section 3 and illustrate the design tasks using several examples. The proposed approach is presented in Section 4 and evaluated in Section 5. Section 6 presents our conclusions.

2. SYSTEM MODEL

2.1. Biochip Architecture

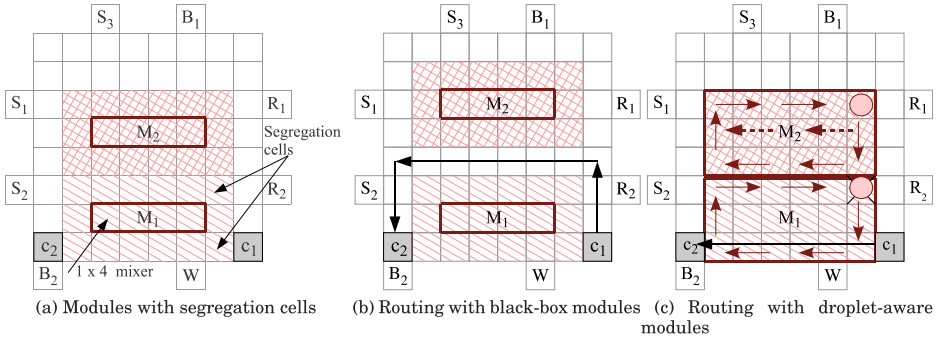
The architecture of a digital biochip is dependent on the actuation mechanism used for creating and manipulating the droplets. The most-used methods are dielectrophoresis (DEP) and electrowetting-on-dielectric (EWOD) [Tabeling 2006]. Both methods are based on electrical forces and can provide high transportation speeds for droplets, using simple biochip architectures [Chakrabarty and Zeng 2007]. In this article we consider digital microfluidic biochips based on the EWOD actuation method.

The schematic of a general EWOD architecture is presented in Figure 1(a). The chip is composed of a microfluidic array of identical cells, together with reservoirs for storing the liquid. Each cell is composed of two parallel glass plates, as shown in Figure 1(b). The top plate contains a single indium tin oxide (ITO) ground electrode, while the bottom plate has several ITO control electrodes. The electrodes are insulated from the droplet through an insulation layer of ParyleneC, on which a thin film of Teflon-AF is added [Srinivasan et al. 2004]. The role of the Teflon layer is to provide a hydrophobic surface on which the droplet will move. The two parallel plates are separated through a spacer, providing a fixed gap height. The droplet moves between the two plates, in a filler fluid (e.g., silicone oil), used in order to prevent evaporation and



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

Fig. 1. Biochip architecture.



(a) Modules with segregation cells (b) Routing with black-box modules (c) Routing with droplet-aware modules

Fig. 2. Microfluidic modules.

the adhesion of molecules on the surface of the chip [Chakrabarty et al. 2010]. Besides the microfluidic array, the chip also contains nonreconfigurable devices, such as detectors and reservoirs, whose locations are fixed after the fabrication of the biochip. The number of nonreconfigurable devices to be integrated on the chip is decided during the design phase.

With EWOD, the droplet is transported on the chip by applying voltages on the control electrodes, thus modifying the contact angle between the liquid and the hydrophobic surface. If the voltage is applied to only one side of the droplet, the gradient in the contact angle at the two edges of the liquid will cause a surface stress in the direction of the applied voltage, leading to the movement of the droplet [Pollack et al. 2002a]. For example, turning off the middle control electrode and turning on the right control electrode in Figure 1(b) will force the droplet to move to the right.

Using the architecture in Figure 1(a), and changing the control voltages, the basic microfluidic operations, such as transport, splitting, dispensing, mixing, and detection, can be performed. For example, two mixing operations on 1×4 modules ($Mixer_1$ and $Mixer_2$)[1] are shown in Figure 2(a). Mixing is done by transporting two droplets to the same location and merging them. Mixing through diffusion, where the resulting droplet remains on the same electrode, is very slow. In order to enhance the mixing process, the droplet is routed over a series of electrodes according to a certain pattern. During movement, the complexity of flow patterns inside the droplet increases, leading

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

to faster operation execution [Paik et al. 2003]. Mixing modules are created by grouping adjacent electrodes on which the droplets corresponding to the operations will be moved. Any cells in the chip can be used for such a purpose, thus we say that the chip is reconfigurable.

Table I presents the results of the experiments performed in Paik et al. [2003], where several mixing times were obtained for various areas, creating a module library. For each experiment, the corresponding droplet has been repeatedly moved inside the virtual module, using a predefined movement pattern. The synthesis tools for biochips proposed so far use such a module library, ignoring the location of the droplet during operation execution and thus considering a module as a black box in which an operation is performed.

However, the position of all droplets on the chip at each time-step is important in order to avoid droplet-merging. If two droplets are next to each other on two adjacent cells, they will tend to merge and form one single droplet. For further details on these fluidic constraints see Section 3. Segregation cells have been used for solving this problem, while ignoring the position of droplets in virtual devices [Su and Chakrabarty 2005; Yuh et al. 2007]. Thus, each device has been surrounded by a 1-cell segregation border to isolate functional areas on which operations are executing (see Figure 2(a)). This results in two segregation cells between modules, although a single cell is enough. The two-cell space has the advantage that is easier to adjust to create droplet routing paths. If the positions of droplets inside devices are not known, routing needs a three-cell path width. So far it has been considered that these paths are created during a post-processing routing step. For example, in Figure 2(b) the position of $Mixer_2$ is modified in order to introduce necessary path for droplet movement between the two mixer modules.

However, segregation cells can be eliminated if we take into account the position of droplets inside modules during execution. Let us consider the two mixers in Figure 2(a). Each mixer is composed of a 1×4 functional area, surrounded by segregation cells to avoid accidental merging. We eliminate the segregation area and consider that the corresponding cells become part of the virtual device (e.g., $Mixer_1$ transforms from a 1×4 to a 3×6 device). We can prevent the accidental merging of the droplets by knowing their locations inside the devices at any time-step. For example, considering the initial positions of the two droplets as shown in Figure 2(c), the mixing operations can be performed by repeatedly routing the droplets according to the movement patterns described by the arrows. The droplets are never too close to each other during execution, so the fluidic constraints are enforced. Such a synchronization of droplets to avoid accidental merging is not always possible.

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

Knowing the locations of droplets inside modules can also be an advantage during the post-synthesis routing step. Let us consider that during the routing step a droplet d must be routed from the cell denoted by c_1 to the cell denoted by c_2 (see Figure 2(b)). The post-synthesis routing algorithms proposed so far have considered devices placed on the chip as obstacles in defining the routes between two modules or between modules and reservoirs, and that the initial placement has to be adjusted in order to introduce the three-cell-width paths necessary for routing, as shown in Figure 2(b). However, droplets can be routed through the functional area of a module, as long as accidental merging is avoided. Let us assume that at time t, the droplets inside the mixers are positioned as shown in Figure 2(c) and are moved according to the pattern shown by the arrows in the mixers. Then droplet d can be routed from the start cell c_1 to the destination cell c_2 on the shortest possible route (shown by the arrow between c_1 and c_2), using electrodes belonging to $Mixer_1$, as long as we ensure

Table I. Module Library

Operation	Area (cells)	Time (s)
Mixing/Dilution	2×4	2.9
Mixing/Dilution	1×4	4.6
Mixing/Dilution	2×3	6.1
Mixing/Dilution	2×2	9.9
Dispensing	–	2

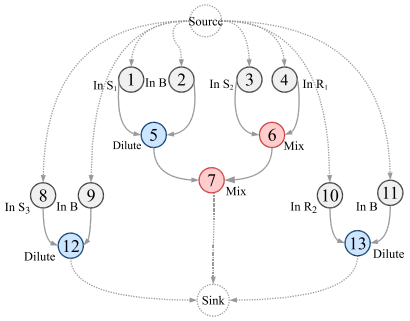


Fig. 3. Application graph.

the fluidic constraints. For example, in order to avoid accidental merging inside $Mixer_1$ we can stop the mixer droplet for four time-steps on its current position (we mark the stopping place by an "X" on the corresponding electrode). This will allow the routed droplet d to be transported on its optimal path to the electrode denoted by c_2. Due to the fact that the droplets in $Mixer_1$ and $Mixer_2$ are no longer synchronized, we cannot continue moving the droplet in $Mixer_2$ according to its original movement pattern, as this would result in an accidental merging with the stopped mixing droplet in $Mixer_1$. Thus, in order to enforce fluidic constraints, we can deviate the movement pattern for the droplet in $Mixer_2$, as shown with dashed arrows in Figure 2(c).

Changing this movement will result in an irregular pattern, and lead to nonstandard operation completion times (i.e., we cannot use the numbers in Table I, which assume a certain fixed movement pattern). Hence, we instead use the execution time calculation method proposed by us in Maftei et al. [2010a] to compute the completion time of an operation on a droplet-aware device.

The analytical method in Maftei et al. [2010a] takes into account the exact movement pattern of a droplet inside a device to give a safe conservative estimate of the operation completion time. We use the routing approach presented in Maftei et al. [2010a] to decide the initial locations of droplets inside modules.

2.2. Biochemical Application Model

We model a biochemical application using an abstract model consisting of a sequencing graph [Chakrabarty and Zeng 2005]. The graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is directed, acyclic, and polar (i.e., there is a *source node*, which is a node that has no predecessors and a *sink node* that has no successors). Each node $O_i \in \mathcal{V}$ represents one operation. The binding of operations to modules in the architecture is captured by the function $\mathcal{B} : \mathcal{V} \rightarrow \mathcal{A}$, where $\mathcal{A}$ is the list of allocated modules from the given library $\mathcal{L}$.

An edge $e_{i,j} \in \mathcal{E}$ from O_i to O_j indicates that the output of operation O_i is the input of O_j. An operation can be activated after all its inputs have arrived and it issues its outputs when it terminates. We assume that for each operation O_i, we know the execution time $C_i^{M_k}$ on module $M_k = \mathcal{B}(O_i)$, where it is assigned for execution. In Figure 3 we

have an example of an application graph with thirteen operations, O_1 to O_{13}. The application consists of two mixing operations (O_6 and O_7), three diluting operations (O_5, O_{12} and O_{13}), and eight input operations (O_1, O_2, O_3, O_4, O_8, O_9, O_{10} and O_{11}). O_5 is a diluting operation during which the concentration of the sample input droplet is changed to an intermediate concentration. The operation is performed by a sequence of mixing and splitting steps. Considering Figure 1(b), a droplet is split by turning on the left and right electrodes and turning off the middle electrode [Ren et al. 2003]. Thus, the droplet volume will vary during the application execution. We assume that the biochemical application has been correctly designed, such that all the operations will have the required input droplet volumes. Let us consider that operation O_5 is bound to a 2×2 diluter module denoted by $Diluter_2$ (i.e., $\mathcal{B}(O_5) = Diluter_2$). Then, according to Table I, the execution time for O_5 will be $C_5^{Diluter_2} = 9.9$ s. The execution[2] of an operation is divided in time-steps of 10 ms, and we capture the set of time-steps with $\mathcal{T}$.

3. PROBLEM FORMULATION

The problem we are addressing in this article can be formulated as follows.

Input.

(1) A biochemical application modeled as a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$;
(2) a biochip consisting of a two-dimensional $m \times n$ array $\mathcal{C}$ of cells;
(3) a characterized module library $\mathcal{L}$;
(4) the maximum number of nonreconfigurable devices of each type that can be integrated on the chip.

Output.

(1) Allocation $\mathcal{A}$, which determines what modules from the library $\mathcal{L}$ should be used;
(2) the binding $\mathcal{B}$ of each operation $O_i \in \mathcal{V}$ to a module $M_k \in \mathcal{A}$;
(3) the schedule $\mathcal{S}$ of the operations, which contains the start time t_i^{start} and finish time t_i^{finish} of each operation O_i on its corresponding module, M_k;
(4) the placement $\mathcal{P}$ containing for each $M_k \in \mathcal{A}$, the bottom left corner $\left(x_k^l, y_k^l\right)$ and the upper right corner $\left(x_k^r, y_k^r\right)$ describing the position of the module on the $m \times n$ array, i.e., $\mathcal{P}(M_k) = \left(x_k^l, y_k^l, x_k^r, y_k^r\right)$;
(5) the route $\mathcal{R}$ taken during the execution of an operation O_i, described by a set of points $\left(x_i^t, y_i^t\right)$, $\forall t \in \left[t_i^{start}, t_i^{finish}\right], \forall O_i \in \mathcal{G}$.

Objective. minimize the schedule length $\delta_{\mathcal{G}}$, i.e., minimize the finishing time t_{sink}^{finish} of the sink node of graph $\mathcal{G}$.

Subject to the following.

(1) Precedence (a) and storage (b) constraints.
   (a) A successor operation can only start executing after the completion of its predecessor: $t_i^{finish} \leq t_j^{start} \forall O_i$ and $\forall O_j \in \mathcal{V}$, such that $\exists e_{i,j} \in \mathcal{E}$.
   (b) If a successor operation is not scheduled immediately after the completion of its predecessor, a storage unit must be placed on the chip:

[2]Approximate time required to route the droplet one cell considering the data in Pollack et al. [2002a]: electrode pitch size = 1.5 mm, gap spacing = 0.3 mm, average linear velocity = 20 cm/s.

$store(O_i, t_{current})$ $\forall t_{current} \in \mathcal{T}$ and $\forall O_i$ and $\forall O_j \in \mathcal{V}$ such that $\exists e_{i,j} \in \mathcal{E}$ and $t_i^{finish} \leq t_{current} < t_j^{start}$.

(2) Nonreconfigurable resource constraints.
No operations bound to the same nonreconfigurable device should overlap in time: $t_i^{finish} \leq t_j^{start}$ $\forall O_i$ and $\forall O_j$ nonreconfigurable operations $\in \mathcal{V}$ such that $\mathcal{B}(O_i) = \mathcal{B}(O_j) = M_k$.

(3) Placement constraints.
No modules placed on the microfluidic array at time t should physically overlap: $x_i^l \leq x_j^r$ or $x_j^l \leq x_i^r$ or $y_i^l \leq y_j^r$ or $y_j^l \leq y_i^r$ $\forall O_i$ and $\forall O_j \in \mathcal{V}$ such that $\mathcal{P}(\mathcal{B}(O_i)) = (x_i^l, y_i^l, x_i^r, y_i^r)$ and $\mathcal{P}(\mathcal{B}(O_j)) = (x_j^l, y_j^l, x_j^r, y_j^r)$ and $t_i^{start} \leq t_j^{start} \leq t_i^{finish}$.

(4) Fluidic constraints during operation execution.
   (a) The location of any two droplets present on the array at time t cannot be directly or diagonally adjacent to each other: $\left|x_i^t - x_j^t\right| \geq 2$ or $\left|x_i^t - y_j^t\right| \geq 2$ $\forall O_i$ and $\forall O_j \in \mathcal{V}$, such that $\mathcal{R}(O_i, t) = (x_i^t, y_i^t)$ and $\mathcal{R}(O_j, t) = (x_j^t, x_j^t)$ $\forall t \in \mathcal{T}$.
   (b) The activated cell for a droplet cannot be adjacent to any other droplet present on the array: $\left|x_i^{t+1} - x_j^t\right| \geq 2$ or $\left|y_i^{t+1} - y_j^t\right| \geq 2$ or $\left|x_i^t - x_j^{t+1}\right| \geq 2$ or $\left|y_i^t - y_j^{t+1}\right| \geq 2$ $\forall O_i$ and $\forall O_j \in \mathcal{V}$, such that $\mathcal{R}(O_i, t) = (x_i^t, y_i^t)$ and $\mathcal{R}(O_j, t) = (x_j^t, y_j^t)$ $\forall t \in \mathcal{T}$.

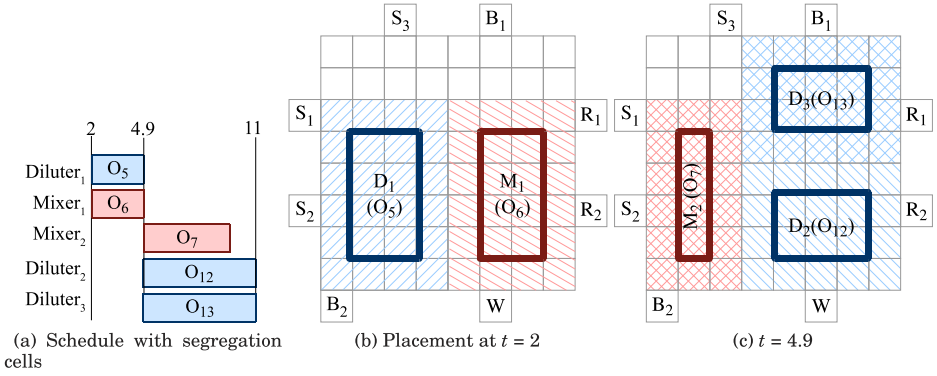
The next sections will illustrate each of the output tasks. The presentation order does not correspond to the order in which our synthesis approach performs these tasks.

3.1. Allocation and Placement

Let us consider the graph shown in Figure 3. We would like to implement the operations on the 8×8 biochip from Figure 1(a). The graph contains two types of operations: nonreconfigurable (input) and reconfigurable (mixing and dilution). The scheduling of input operations is determined at the same time as the other operations, the fixed number of reservoirs representing a constraint to the final completion time of the application. However, inputs execute outside the microfluidic array and therefore do not affect the placement of the other operations. We assume that the locations of reservoirs have been decided during the fabrication of the chip and are as shown in Figure 1(a). We need to assign each input operation to a reservoir of the same type, e.g., O_2 can only be assigned to one of the buffer reservoirs, B_1 and B_2. Let us consider that the input operations are assigned to the input ports as follows: O_1 to the input port S_1, O_2 to B_1, O_3 to S_2, O_4 to R_1, O_8 to S_3, O_9 to B_1, O_{10} to R_2, and O_{11} to B_2. The synthesis approach will have to decide the scheduling of the input operations and make sure that each reservoir is used by at most one input operation at each time-step. For the reconfigurable operations in Figure 3, the mixing operations (O_6 and O_7) and the dilution operations (O_5, O_{12} and O_{13}), our synthesis approach will have to allocate the appropriate modules, bind operations to them, and perform the placement and scheduling.

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

One solution to the problem is presented in Figure 4, where the following modules are used: one 2×4 mixer (4×6 with segregation area), one 2×4 diluter (4×6 with segregation area), one 1×4 mixer (3×6 with segregation area), and two 2×3 diluters (4×5 with segregation area). The resulting schedule for this allocation is shown in Figure 4(a). The schedule is depicted as a Gantt chart, where for each module, we



(a) Schedule with segregation cells (b) Placement at $t = 2$ (c) $t = 4.9$

Fig. 4. Black-box operation execution example.

represent the operations as rectangles with their length corresponding to the duration of that operation on the module.

The placement for the allocation and schedule is as indicated in Figures 4(b)–(c). Our placement problem has similarities with the placement of DR-FPGAs, where virtual modules can physically overlap on-chip as long as they do not overlap in time, i.e., they are used during different time intervals. After an operation has finished executing on a module, we can reuse the same cells as part of another module.

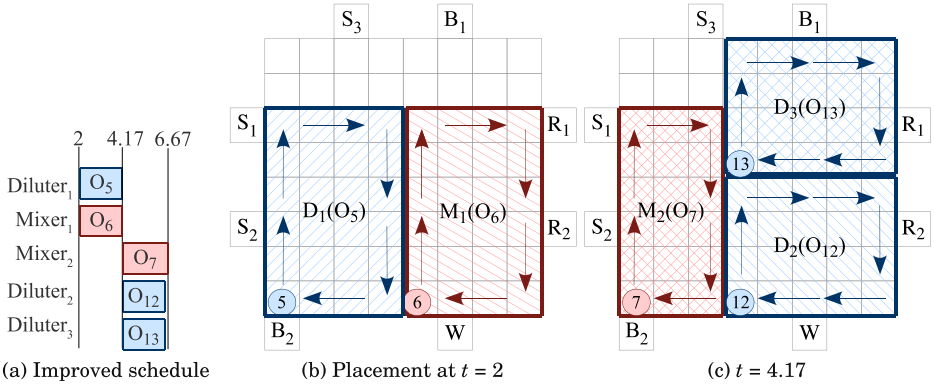
The placement problem of DMBs can also include finding the location of nonreconfigurable devices (e.g., reservoirs, optical detectors), whose number is constrained by the design specifications. As input operations are executed outside the microfluidic array, the positions of reservoirs are determined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

3.2. Binding and Scheduling

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

Considering the graph in Figure 3 and the allocation presented in Section 3.1, Figure 4(a) presents the optimal schedule in the case when we do not consider the position of droplets inside the virtual modules. For example, operation O_7 is bound to module $Mixer_2$, starts immediately after the diluting operation O_5 $\left(t_7^{start} = 4.9\right)$ and takes 4.6 s, finishing at time $t_7^{finish} = 9.5$ s. We consider that input operations are scheduled for execution as follows: $t_1^{start} = t_2^{start} = t_3^{start} = t_4^{start} = 0$ s, $t_8^{start} = t_9^{start} = t_{10}^{start} = t_{11}^{start} = 2.9$ s. Each dispensing operation takes 2 s, as shown in Table I. For space reasons, we do not show the schedule of input operations, however the starting times of the reconfigurable operations shown in Figure 4(a) do take into consideration the time required for dispensing the droplets on the microfluidic array.

Note that special "store" modules have to be allocated if a droplet has to wait before being processed, which is different from DR-FPGAs. In general, if there exists an edge $e_{i,j}$ from O_i to O_j such that O_j is not immediately scheduled after O_i (i.e., there is a delay between the finishing time of O_i and the start time of O_j), then we will have to



(a) Improved schedule (b) Placement at $t = 2$ (c) $t = 4.17$

Fig. 5. Droplet-aware operation execution example.

allocate a storage cell for $e_{i,j}$. Hence, the allocation of storage cells depends on how the schedule is constructed. Any available cell on the microfluidic array can be used for temporarily storing the droplet.

3.3. Synthesis with Droplet-Aware Operation Execution

The schedule presented in Figure 4(a) is optimal for the given allocation considering that the positions of droplets inside modules are unknown during operation execution. Therefore, modules are surrounded by segregation cells, which ensure that the fluidic constraints are satisfied at each time-step. However, the solution can be further improved (see Figure 5(a)) by taking into account the location of droplets inside virtual modules. Consider the same synthesis example as in Section 3.1, with the allocation presented in Figure 4(a). At time $t = 2$, operations O_5 and O_6 are scheduled, and modules $Diluter_1$ and $Mixer_1$ are placed on the chip. Let us assume that the droplets corresponding to the two operations are routed to the positions shown in Figure 5(b), where the dilution and mixing operations start executing, according to the illustrated movement patterns.

We eliminate the segregation cells, and consider them as part of the functional area of the devices. For example, operation O_5, which was initially bound to a 2×4 device can now be executed by routing the corresponding droplet on a 4×6 area. The area occupied for performing O_5 remains the same as in Section 3.1, however all the cells in the device can now be used for operation execution. By routing the droplets corresponding to O_5 and O_6 as shown in Figure 5(b), the droplets are never too close and therefore the fluidic constraints are enforced. The same situation is shown in Figure 5(c), where operations O_7, O_{12}, and O_{13} are repeatedly routed from their initial positions according to the depicted movement patterns, without the need of segregation cells.

The completion times for the droplet-aware operations shown in Figure 5 are computed using the analytical method proposed by us in Maftei et al. [2010a]. Although for simplicity reasons, the movement patterns of the droplets in Figure 5 are synchronized, this is not always possible due to fluidic constraints. Our approach takes this into consideration by allowing a flexible movement pattern of the droplets during operation execution. In order to avoid accidental merging, a droplet can be deviated from its preestablished movement pattern according to the characterized module library, or can be kept at the same location on the chip for several time-steps.

The exact routes taken by droplets inside a module during operation execution are determined offline and are stored in the memory of a microcontroller, which coordinates the activation of the electrodes on the microfluidic array. In order to minimize

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

Fig. 6. Synthesis algorithm for DMBs.

memory requirements we consider that only the preestablished routes and the deviations of the droplets from these routes will be recorded in memory, in a compressed form.

4. TABU SEARCH-BASED SYNTHESIS

The problem presented in the previous section is NP-complete. Scheduling in even simpler contexts is NP-complete [Ullman 1975]. In addition, the placement is equivalent to a sequence of 2D packing problems, known to be NP-complete [Garey and Johnson 1979].

Our synthesis strategy, presented in Figure 6, takes as input the application graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the given biochip cell array $\mathcal{C}$, and the module library $\mathcal{L}$, and produces that implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P}, \mathcal{R} >$ consisting of, respectively, the allocation $\mathcal{A}$, binding $\mathcal{B}$, scheduling $\mathcal{S}$, placement $\mathcal{P}$, and routing $\mathcal{R}$ of operations during execution, which minimizes the schedule length $\delta_{\mathcal{G}}$ on the given biochip $\mathcal{C}$. In this article, we use a Tabu Search (TS) metaheuristic to decide the allocation $\mathcal{A}$ and binding $\mathcal{B}$ (line 3 in Figure 6). For a given allocation and binding decided by TS, we use a List Scheduling (LS) heuristic [Micheli 1994] to decide the schedule $\mathcal{S}$ of the operations. As the result of the synthesis process depends on the order of executing the operations, we use priorities Π to decide the scheduling sequence for two or more operations that are ready to be executed at the same time t.

4.1. List Scheduling

For given allocation, binding and priorities decided by TS, we use the *ScheduleAndPlace* function in Figure 7 to determine the scheduling $\mathcal{S}$, placement $\mathcal{P}$, and routing $\mathcal{R}$ of droplets inside modules, during operation execution. Our scheduling is based on a List Scheduling heuristic. LS takes as input the application graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the cell array $\mathcal{C}$, the module library $\mathcal{L}$, the allocation $\mathcal{A}$, binding $\mathcal{B}$, and priorities Π, and returns the scheduling $\mathcal{S}$, placement $\mathcal{P}$, and the routes of droplets inside modules $\mathcal{R}$. The List Scheduling heuristic is based on a sorted priority list, L_{ready}, containing the operations $O_i \in \mathcal{V}$, which are ready to be scheduled. TS starts from an initial solution, where we consider that each operation $O_i \in \mathcal{V}$ is bound to a randomly chosen module $\mathcal{B}(O_i) \in \mathcal{L}$ (line 1 in Figure 6). The initial execution priorities, Π°, are given according to the bottom-level values of the nodes in the graph (line 2) [Sinnen 2007]. According to this, the priority of an operation is defined as the length of the longest path from the operation to the sink node of the graph. The start and finish times of all the operations are initialized to 0 in the beginning of the algorithm (lines 2 and 3 in Figure 7). A list $L_{execute}$, which contains the operations that are executing at the current time step is created in the beginning of the algorithm (line 4). Initially, L_{ready} will contain those operations in the graph that do not have any predecessors (line 5 in Figure 7). We do not consider input operations as part of the ready list. As they do not have any precedence constraints, input operations can be executed at any moment. However, it is important that inputs and their successors are performed sequentially, in order to avoid storing the dispensed droplets. Let us consider moment $t_{current}$ during the execution of the application. For all the operations that finish executing at $t_{current}$ we

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

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

Fig. 7. List scheduling algorithm for DMBs.

check if their successors are ready to be scheduled (line 14 in Figure 7). An operation is considered to be ready if all its predecessors (except input operations) have finished executing. Next, we try and schedule the ready operations, starting with operation O_j having the highest priority (line 17 in Figure 7). Before O_j can be scheduled, its input constraints must be checked. If O_j has as predecessor an input operation O_k, we try to schedule O_k such that $t_k^{finish} = t_j^{start} = t_{current}$. However, as reservoirs/dispensing ports are nonreconfigurable devices, their number is constrained during design specifications. That is, operation O_j can be scheduled at time t only if at time $t - C_k^{reservoir}$ there is an available reservoir/dispensing port on which O_k can be executed. Otherwise, O_j will be delayed and the next highest priority operation is considered for execution. If all the constraints related to O_j are satisfied, its corresponding module, $\mathcal{B}(O_j)$, is placed on the microfluidic array (line 18 in Figure 7) and the start time of the operation is updated (line 21). If there exists a placed storage module associated with the operation O_j, the storage is removed and the placement is updated.

The combined scheduling, placement, and routing during operation execution is implemented by the *ScheduleAndPlace* function (Figure 7). Once an operation is scheduled it is removed from L_{ready} and added to $L_{execute}$. Before the end of the iteration, the storage constraints are considered. For all the operations that finished at $t_{current}$, the placement of the microfluidic array must be updated by removing the modules to

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

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

Fig. 8. Placement algorithm for DMBs.

which they are bound (line 11 in Figure 7). Also, if their successors have not yet been scheduled for execution, a storage unit is placed on the microfluidic array. TS uses design transformations to search the solution space. Inside TS, we use the *Schedule-AndPlace* function to determine the schedule, placement and routing of droplets during operation execution for an implementation Ψ.

The next section presents our proposed placement algorithm, while Sections 4.3 and 4.4 present our droplet-aware operation execution algorithm and TS implementation, respectively. Section 4.5 discusses the time complexity of the proposed TS algorithm.

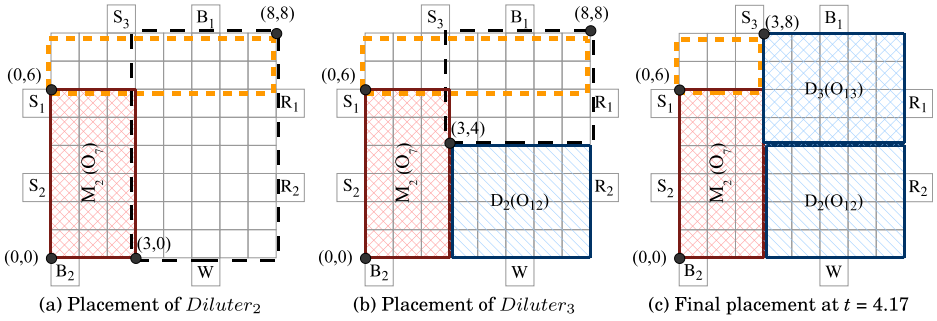
4.2. Placement Algorithm

The placement for DMBs can be considered as a 2D rectangle packing problem, in which at each time-step, the position of modules to be accommodated on the microfluidic array must be decided. Our placement approach, presented in Figure 8, is based on the "keep all maximal empty rectangles" (KAMER) algorithm proposed in Bazargan et al. [2000] for DR-FPGAs. The algorithm partitions the free space on the chip into a list of overlapping rectangles, represented by the coordinates of their left bottom and right upper corners. When an operation is scheduled, the *Placement* algorithm in Figure 8 selects an empty rectangle that best fits the module to which the operation is bound. If there is no empty rectangle that can accommodate the device, the corresponding operation will be delayed until more space is freed on the microfluidic array.

Consider the synthesis example in Figure 5(c). At t = 4.17 s there are three operations that are ready to be scheduled, hence $L_{ready} = \{O_7, O_{12}, O_{13}\}$. We consider that the reconfigurable operations are bound to the same devices as in Section 3.3, thus O_7 is bound to a 3 $\times$ 6 mixer, O_{12}, and O_{13} to 4 $\times$ 5 diluters. Then the priorities of the operations ready to be scheduled are computed based on the optimal routes on which the operations can be executed inside the modules: $\pi_{O_7} = C_{O_7}^{Mixer_2} = 2.5$, $\pi_{O_{12}} = C_{O_{12}}^{Diluter_3} = 2.25$, $\pi_{O_{13}} = C_{O_{13}}^{Diluter_4} = 2.25$.

Accordingly, the LS algorithm will select O_7 and will call *Placement* to place $Mixer_2$ on the biochip array. At time t = 4.17 s, mixing operation O_5 and dilution operation O_6 have finished executing, therefore the microfluidic array forms one big empty rectangle $Rect_1 = (0, 0, 8, 8)$. The mixer bound to O_7 is placed at the bottom corner of $Rect_1$ (line 6 in Figure 8). Consequently, in line 7, the free space will be updated to $L_{rect} = \{Rect_1 = (3, 0, 8, 8), Rect_2 = (0, 6, 8, 8)\}$, as depicted in Figure 9(a).

After the scheduling and placement of O_7, the next operation to be considered for scheduling is O_{12}. As rectangle $Rect_1 = (3, 0, 8, 8)$ is the only one sufficiently large to accommodate the 4 $\times$ 5 module (line 4), $Diluter_2$ will be placed at its bottom corner and the free space will be updated to $L_{rect} = \{Rect_1 = (3, 4, 8, 8), Rect_2 = (0, 6, 8, 8)\}$, see



(a) Placement of $Diluter_2$ (b) Placement of $Diluter_3$ (c) Final placement at t = 4.17

Fig. 9. Placement example.

Figure 9(b). Finally, $Rect_1$ will be chosen for accommodating the diluter bound to O_{13} and the final placement at t = 4.17 will be as shown in Figure 9(c).

In this case, the free space on the microfluidic array permits scheduling O_7, O_{12}, and O_{13} at time t. If however, there is not enough space to place the module bound to a ready operation, the scheduling of the operation will have to be delayed.

4.2.1. Placement of Nonreconfigurable Devices. The placement of a nonreconfigurable device (e.g., an optical detector) on the microfluidic array is similar to that of a reconfigurable module. However, once decided, the location of the device remains fixed throughout the execution of the application. Therefore our algorithm maintains a list of locations at which nonreconfigurable operations of each type (e.g., detection operations) can be performed, $L_{nonReconf}$. These locations are established during the execution of the placement algorithm. The size of the list is constrained by the maximum number of devices of the given type that can be integrated on the chip, given as an input during design specifications. Let us consider that at time t a nonreconfigurable detection operation is ready to be scheduled. We try to place the 3×3 detector at one of the locations in L_{detect}. If no locations have been established previously or if they are all occupied but we can still integrate detectors on the array, we use the algorithm in Figure 8 to find a new detector location. If a free rectangle that can accommodate the 3×3 module is found, the operation is scheduled at time t and the point corresponding to the left bottom corner of the rectangle is added to L_{detect}. Otherwise the detection operation cannot be scheduled at time t. Just as in the case of reconfigurable modules, nonreconfigurable devices cannot overlap with other modules placed on the chip.

4.3. Droplet-Aware Operation Execution

The movement of droplets during operation execution is determined offline by successively calling the *RunOperationsOneTimeStep* algorithm presented in Figure 10. The algorithm takes as input the list of operations executing at $t_{current}$, $L_{execute}$, the $m \times n$ matrix $\mathcal{C}$ of cells, the current placement of modules $\mathcal{P}$, the partial routes $\mathcal{R}$ of droplets inside devices up to time $t_{current}$, the module library $\mathcal{L}$, and the current time-step $t_{current}$. For each operation O_i under execution at $t_{current}$, the algorithm decides the movement of the corresponding droplet inside the module $M_k = B(O_i)$ to which the operation is bound, for the next time-step. Compared to previous approaches, we consider that the movement pattern followed by a droplet during operation execution can be dynamically changed, in order to ensure fluidic constraints, and at the same time minimize the completion time of the operation.

The analytical method proposed in Maftei et al. [2010a] is used for characterizing the execution of operations, using as a starting point a given module library. According

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

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

Fig. 10. Droplet-aware operation execution algorithm for DMBs.

to this method, any route can be decomposed into a sequence of forward, backward, and perpendicular moves. In order to determine the completion time of an operation following an irregular movement pattern, we need to approximate the percentage of execution performed over one cell, corresponding to each type of move. The method proposed in Maftei et al. [2010a] provides safe estimates of completion percentages, by decomposing the modules in the given module library that have preestablished movement patterns and known completion times, determined through experiments. As a result, the method can be used to approximate the amount of operation completion for any given droplet, during operation execution.

Let us consider the example in Figure 11(a), at time $t_{current}$. There are three operations executing on the array: O_7, bound to a 3 $\times$ 6 mixer module and O_{12} and O_{13}, bound to 4 $\times$ 5 diluters. Let us consider that the previous three moves for the operations are as indicated in Figure 11(b), by the position of the droplets, and the corresponding connecting arrows. We use a greedy approach for deciding the directions in which the droplets are moved *at the current time-step*. For each droplet we have a number of feasible moves that can be performed, while avoiding accidental merging. We consider that the quality of each move is given by the amount of operation completion performed while transporting the droplet in the corresponding direction. We use the analytical method proposed by us in Maftei et al. [2010a] to evaluate the quality of each move (lines 2–4 in Figure 10). Consequently, according to our greedy approach, the droplet is transported in the best direction (line 6) and the percentage of operation completion is updated (line 7), using the method proposed in Maftei et al. [2010a]. If the operation finished executing (its completion percentage reached 100%) then its finishing time is also updated (lines 8–10).

Let us consider that the first droplet to be moved in Figure 11 is the one corresponding to mixing operation O_7. The droplet can be moved downwards, backwards, or it can remain at the current position, see Figure 11(c). Based on the droplet characterization in Maftei et al. [2010a], the droplet is routed downwards, as this leads to the most mixing out of the feasible moves. After O_7 is routed, the next droplet to be moved is O_{12}. The droplet cannot continue its movement upwards, as it risks accidentally merging with O_7 (see Figure 11(d)). Hence, as shown in Figure 11(e), O_{12} is transported to the right compared to its current position, which is the best possible move. Finally, the algorithm chooses to keep O_{13} on the current position, as moving it backwards leads to negative mixing [Maftei et al. 2010a] and moving it downwards breaks the fluidic constraints (accidental merging with O_7). Figure 11(f) shows the positions of the droplets at time $t_{current}$ + 1, after the moves have been performed.

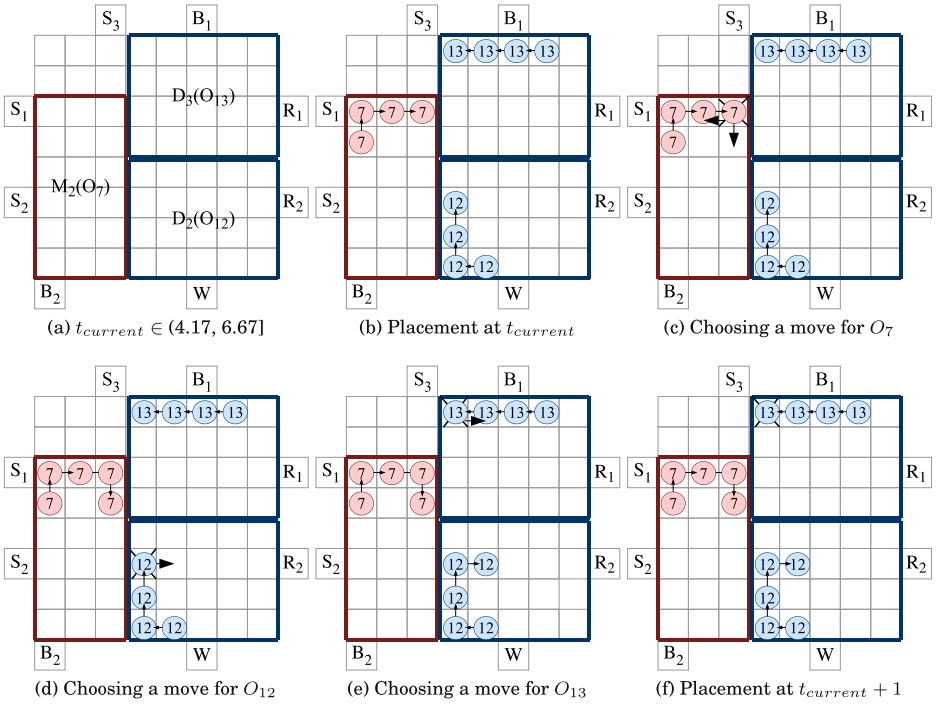


Fig. 11. Running operations O_7, O_{12}, and O_{13} for one time-step.

4.4. Tabu Search

Tabu Search (TS) [Glover and Laguna 1997] is a metaheuristic method used for solving optimization problems, by incorporating memory structures during the search process. This method has already been used successfully for a wide range of optimization problems, such as scheduling, research planning, and VLSI design [Glover and Laguna 1997]. Tabu Search is based on a neighboring technique, using design transformations (moves) applied to the current solution, $\Psi^{current}$, to generate a set of neighboring solutions, N, that can be further explored by the algorithm.

In order to efficiently explore the search space and to escape local optimality, the metaheuristic uses a memory structure that records the recently visited solutions (a tabu list). The information related to the last performed moves is used to guide Tabu Search through the search space, by restricting the possibility of reversing a previously visited solution. However, labeling a move as tabu can result in prohibiting attractive solutions that have not been visited so far. In order to prevent this situation, an "aspiration criteria" can be used, which allows a tabu solution to be visited, if it improves on the currently best known one. Moreover, two other strategies called "diversification" and "intensification" can be integrated in the Tabu Search in order to improve the search process. During diversification, the metaheuristic is encouraged to explore previously unvisited regions of the search space, and thus incorporate new elements that were not previously included in the solution. The intensification strategy focuses on the extensive exploration of promising regions of the search space that have already been visited, looking to improve the best found solution.

Our Tabu Search-based algorithm for droplet-aware module-based synthesis is presented in Figure 12. In order to explore the search space the algorithm uses two types of moves: (1) rebinding moves; and (2) priority swapping moves. During a rebinding

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

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

Fig. 12. Tabu Search algorithm for DMBs.

move an operation $O_i \in \mathcal{G}$ is randomly selected and its binding is changed to a different device in the module library. A priority swapping move consists in swapping the priorities of two randomly chosen operations in the graph. We have used priority swapping as part of a diversification strategy, when the best found solution does not improve for a number of iterations, num_{div}, determined experimentally.

We have considered that for each type of move, our algorithm maintains a tabu list, consisting of the recently performed transformations. In order to reduce the amount of memory required to memorize the search history, we do not record the entire solutions, but only the attributes that have changed as part of a transformation. For example, if operation O_i is rebound to module M_j as part of a rebinding move, the transformation will be recorded in the corresponding tabu list as a pair of the form (O_i, M_j). The algorithm starts with an initial solution Ψ°, where each operation is bound to a randomly chosen module and has a priority given according to the bottom-level value of the node corresponding to the operation. In order to evaluate the quality of the solution, the *ScheduleAndPlace* function is used, which returns the schedule $\mathcal{S}^\circ$ and placement $\mathcal{P}^\circ$ for the given allocation and binding (line 1).

Two empty tabu lists, $tabuList_{dev}$ and $tabuList_{prio}$, used to record the rebinding moves and the priority swapping moves, respectively, are initialized in lines 4–5. Each list has a given size, $tabuSize_{dev}$ and $tabuSize_{prio}$ correspondingly, specifying

the maximum number of moves that can be recorded. In order to implement the diversification strategy, we use a variable num_{iter}, which keeps track of the number of iterations passed without the improvement of the best solution, $\Psi_{\mathcal{G}}^{best}$ (line 6). While the time limit set for running the algorithm has not been reached, we try to improve the best found solution, by using a number of iterations (lines 7–29). During each iteration, we perform rebinding moves to the current solution, and construct a set of neighboring solutions, N (line 8). However, N might contain solutions that are disallowed by TS. For example, according to the aspiration criteria, if a move labeled as tabu is performed, the resulting solution is allowed only if it improves on the currently best known solution. Therefore, all the tabu moves that are leading to a worse completion time than the best one, are removed from N and the set $\tilde{N}$ of allowed moves is created (line 9). These moves are evaluated using the *ScheduleAndPlace* function, and the one leading to the best schedule is selected and marked as tabu (lines 10–12). If the obtained solution has a better schedule length that the currently known one, the best-so-far solution is updated (lines 14–15). However, if the best known solution does not improve for a given number of iterations num_{div}, our algorithm introduces diversification into the search by performing a priority swapping move (line 18). If as a result of diversification the best move is improved, then Ψ^{best} is updated to $\Psi^{current}$ (line 23). After diversification is performed, the variable num_{iter} is reset to 0 (line 26).

The Tabu Search-based algorithm in Figure 12 is given a time limit, during which it repeatedly performs moves trying to find the best solution in the search space. When the time limit is reached the best found solution in terms of schedule length, Ψ^{best}, is returned by the algorithm, which then terminates.

4.5. Time Complexity Analysis

Let us consider the overall algorithm presented in Figure 6. The time complexity is given by the *Tabu Search* algorithm in line 3, which uses the *ScheduleAndPlace* algorithm presented in Figure 7 to perform the scheduling, placement, and droplet-aware operation execution for all the operations O_i in $\mathcal{V}$. In order to implement the placement algorithm in Figure 8 we use the area matrix data structure proposed in Handa and Vemuri [2004]. According to this, the microfluidic array is modeled as a two dimensional array m $\times$ n, in which each cell represents an electrode and stores a value. The value can be either positive, giving the number of contiguous empty cells above the cell, in the same column, or negative, if the cell is occupied by a module. The data structure leads to an efficient management of the free space using overlapping rectangles, requiring $O(mn)$ for inserting a new module and the same for deleting a module from the microfluidic array. As the placement is performed for all the ready operations (line 18 in Figure 7), it has complexity $O(|\mathcal{V}|\,mn)$. If we consider the while loop at line 7 in Figure 7, it contains two loops, the first one executing for operations that are finishing at the current time-step and having complexity $O(|\mathcal{V}|\,mn)$ and the second one, for placing ready operations, also requiring $O(|\mathcal{V}|\,mn)$. In order to perform the execution of operations, the *ScheduleAndPlace* uses the algorithm in Figure 10, which decides for each droplet on the array, which of the maximum five movements is the best one to be performed. To ensure the fluidic constraints and therefore define the valid moves for a droplet, the locations of all other droplets present on the chip must be considered (line 3 in Figure 10). As a result, the *RunOperationsOneTimeStep* algorithm has a complexity of $O(|\mathcal{V}|^2)$. Considering that the number of elements in the neighborhood equals the number of operations in $\mathcal{V}$, the overall complexity of the Tabu Search algorithm is $O(\left|\mathcal{V}^3\right| mn)$.

Table II. Best-, Average Schedule Length, and Standard Deviation for the Real-Life Applications

Application	Area (cells × cells)	Best (s)		Average (s)		Standard dev. (%)	
		DAS	MBS	DAS	MBS	DAS	MBS
In-vitro	8 × 9	69.83	70.40	72.41	75.72	1.86	3.01
	8 × 8	71.69	82.43	83.67	91.31	11.73	9.63
	7 × 8	74.13	86.82	82.93	95.73	8.01	8.69
Proteins	15 × 15	96.60	102.20	99.66	112.22	1.07	4.63
	14 × 14	95.63	107.12	99.68	116.78	1.12	5.34
	13 × 13	98.76	117.25	101.00	128.75	0.65	6.46

5. EXPERIMENTAL EVALUATION

In order to evaluate our droplet-aware operation execution approach, we have used two real-life applications and three synthetic TGFF-generated benchmarks. The Tabu Search algorithm was implemented in Java (JDK 1.6), running on an Intel Core i7 860 at 2.8 GHz with 8 GB of RAM. The droplet movement characterization of operation execution is based on the decomposition of devices shown in Table I, using the analytical method proposed in Maftei et al. [2010a].

In our experiments we were interested to determine the improvement in completion time that can be obtained by eliminating segregation cells and considering the position of droplets inside devices. Therefore we consider two approaches to the synthesis problem: a droplet-aware operation execution approach (Droplet-Aware Synthesis, DAS) and a black-box operation execution approach (Module-Based Synthesis, MBS). For MBS, we have used the synthesis method we proposed in Maftei et al. [2010b].

In order to determine the initial positions of droplets inside modules during droplet-aware operation execution, we have used the GRASP method developed by us in Maftei et al. [2010a].

Table II presents the results obtained by using DAS and MBS for the synthesis of two real-life applications: in-vitro diagnostics on human physiological fluids [Su et al. 2006], which has 28 operations, and the colorimetric protein assay (103 operations) [Su and Chakrabarty 2005]. Column 3 in the table represents the best solution out of 50 runs (in terms of the application completion time $\delta_{\mathcal{G}}$) for the droplet-aware approach, and the black-box approach. The average and standard deviation over the 50 runs compared to the best application completion time are also reported in Table II. The comparison is made for three progressively smaller areas. In Maftei et al. [2010b] we have shown that the quality of solutions produced by the MBS implementation does not degrade significantly if we reduce the time limit from 60 minutes to 10 minutes. Hence, we have decided to use a time limit of 10 minutes for all the experiments in this article. A fast exploration is important since we envision using DAS for architecture exploration, where several biochip architectures have to be quickly evaluated in the early design phase (considering not only different areas, but also different placement of nonreconfigurable resources).

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

Table III. Best-, Average Schedule Length, and Standard Deviation for the Synthetic Benchmarks

Operations	Area (cells × cells)	Best (s)		Average (s)		Standard dev. (%)	
		DAS	MBS	DAS	MBS	DAS	MBS
20	8 × 8	40.99	45.01	41.79	47.63	0.80	2.01
	7 × 8	41.32	45.75	43.15	50.46	0.98	2.64
	7 × 7	42.15	47.81	46.23	56.77	1.50	6.14
40	9 × 10	46.85	49.60	47.25	53.93	0.17	2.58
	9 × 9	47.38	51.10	47.76	55.49	0.25	2.60
	8 × 8	47.47	83.83	55.16	92.35	12.27	4.47
60	9 × 10	82.69	84.00	84.88	89.07	1.26	3.11
	9 × 9	82.40	85.43	85.27	95.14	1.40	5.02
	8 × 9	87.54	100.56	95.87	111.89	4.19	7.18

A measure of the quality of a TS implementation is how consistently it produces good quality solutions. The results shown in Table II were obtained for 50 runs of the DAS and MBS approaches. The standard deviations over the 50 runs compared to the best application completion times $\delta_{\mathcal{G}}$ are reported in column 5. We can notice the standard deviation with DAS is small, which indicates that DAS consistently finds solutions that are very close to the best solution found over the 50 runs (each run will explore differently the solution space, resulting thus in different solutions).

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

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

6. CONCLUSIONS

In this article we have presented a Tabu Search-based technique for the synthesis of digital microfluidic biochips. Compared to previous approaches, we have considered the positions of droplets inside virtual devices, during their execution. Two real-life examples as well as a set of three synthetic applications have been used for evaluating the effectiveness of the proposed approach. We have shown that by considering the locations of droplets inside devices we can better utilize the area of the microfluidic array, leading to improvements in the completion times of applications as compared to the black-box approach. By reducing the execution time, smaller area biochips can be used, resulting in a decrease of the design costs necessary for running the biochemical applications.

REFERENCES

Bazargan, K., Kastner, R., and Sarrafzadeh, M. 2000. Fast template placement for reconfigurable computing systems. *IEEE Des. Test Comput. 17*, 1, 68–83.

Chakrabarty, K. 2010. Design automation and test solutions for digital microfluidic biochips. *IEEE Trans. Circuits Syst. I, Reg. Papers 57*, 4–17.

Chakrabarty, K., Fair, R. B., and Zeng, J. 2010. Design tools for digital microfluidic biochips: Towards functional diversification and more than Moore. *Trans. Computer-Aided Design Integr. Circuits Syst. 29*, 7, 1001–1017.

Chakrabarty, K. and Zeng, J. 2005. Design automation for microfluidics-based biochips. *ACM J. Emerg. Technol. Comput. Syst. 1*, 3, 186–223.

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

Fair, R. B. 2007. Digital microfluidics: Is a true lab-on-a-chip possible? *Microfluidics Nanofluidics 3*, 3, 245–281.

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

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

Handa, M. and Vemuri, R. 2004. An efficient algorithm for finding empty space for online FPGA placement. In *Proceedings of the Design Automation Conference*. 960–965.

ITRS07. International Technology Roadmap for Semiconductors. http://www.itrs.net/Links/2007ITRS/Home2007.htm.

Maftei, E. 2011. Synthesis of digital microfluidic biochips with reconfigurable operation execution. Ph.D. thesis, Technical University of Denmark.

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

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

Maftei, E., Paul, P., and Madsen, J. 2010b. Tabu search-based synthesis of digital microfluidic biochips with dynamically reconfigurable non-rectangular devices. *J. Des. Autom. Emb. Syst. 14*, 287–308.

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

Micheli, G. D. 1994. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill Science.

Moore, G. E. 1965. Cramming more components onto integrated circuits. *Electronics 38*, 8, 114–117.

Paik, P., Pamula, V. K., and Fair, R. B. 2003. Rapid droplet mixers for digital microfluidic biochips. *Lab Chip 3*, 253–259.

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

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

Ricketts, A., Irick, K., Vijaykrishnan, N., and Irwin, M. 2006. Priority scheduling in digital microfluidics-based biochips. In *Proceedings of Design, Automation and Test in Europe*. Vol. 1. 1–6.

Sinnen, O. 2007. *Task Scheduling for Parallel Systems.* Wiley.

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

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

Su, F. and Chakrabarty, K. 2005. Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips. In *Proceedings of the 42nd Annual Conference on Design Automation.* 825–830.

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

Tabeling, P. 2006. *Introduction to Microfluidics*. Oxford University Press.

Ullman, D. 1975. NP-complete scheduling problems. *J. Comput. Syst. Sci. 10*, 384–393.

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

Yuh, P.-H., Yang, C.-L., and Chang, Y.-W. 2007. Placement of defect-tolerant digital microfluidic biochips using the T-tree formulation. *ACM J. Emerg. Technol. Comput. Syst. 3*, 3.

Received October 2010; revised April 2011, August 2011; accepted November 2011