Tabu Search-Based Synthesis of Dynamically Reconfigurable Digital Microfluidic Biochips

Elena Maftei
em@imm.dtu.dk

Paul Pop
pop@imm.dtu.dk

Jan Madsen
jan@imm.dtu.dk

DTU Informatics
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark

ABSTRACT

Microfluidic biochips are replacing the conventional biochemical analyzers, and are able to integrate on-chip all the necessary functions for biochemical analysis. The "digital" microfluidic biochips are manipulating liquids not as a continuous flow, but as discrete droplets, and hence they are highly reconfigurable and scalable. A digital biochip is composed of a two-dimensional array of cells, together with reservoirs for storing the samples and reagents. Several adjacent cells are dynamically grouped to form a virtual device, on which operations are executed. During the execution of an operation, the virtual device can be reconfigured to occupy a different group of cells on the array. In this paper, we present a Tabu Search metaheuristic for the synthesis of digital microfluidic biochips, which, starting from a biochemical application and a given biochip architecture, determines the allocation, resource binding, scheduling and placement of the operations in the application. In our approach, we consider moving the modules during their operation, in order to improve the completion time of the biochemical application. The proposed heuristic has been evaluated using three real-life case studies and ten synthetic benchmarks.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Performance, Design

Keywords

Microfluidics, biochips, reconfigurability

1. INTRODUCTION

Microfluidic biochips (also referred to as lab-on-a-chip) represent a promising alternative to conventional biochemical laboratories, and are able to integrate on-chip all the necessary functions for biochemical analysis using microfluidics, such as, transport, splitting, merging, dispensing, mixing, and detection [8].

CASES'09, October 11–16, 2009, Grenoble, France.

Biochips offer a number of advantages over conventional biochemical procedures. By handling small amount of fluids, they provide higher sensitivity while decreasing reagent consumption, hence reducing cost. Moreover, due to their miniaturization and automation, they can be used as point-of-care devices, in areas that lack the infrastructure needed by conventional laboratories [20].

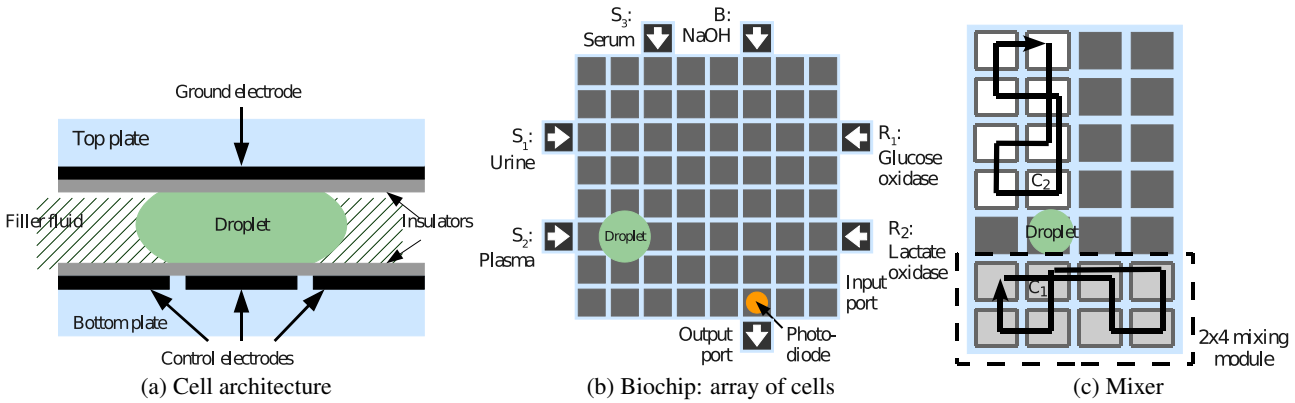
Due to these advantages, biochips are expected to revolutionize clinical diagnosis, especially immediate point of care diagnosis of diseases. Other emerging application areas include drug discovery, DNA analysis (e.g., polymerase chain reaction and nucleic acid sequence analysis), protein and enzyme analysis and immuno-assays. Microfluidic devices can also be used for environment monitoring, by pathogen detection in air or water samples [20].

There are two generations of microfludic biochips. The first generation is based on the manipulation of continuous liquid through fabricated micro-channels, using external pressure sources or integrated mechanical micro-pumps [20]. Although adequate for many simple biochemical applications, their integrated micro-structures make continuous-flow biochips unsuitable for more complex applications, requiring complicated fluid manipulations [3]. The second generation is based on the manipulation of discrete, individually controllable droplets, on a two-dimensional array of identical cells. The actuation of droplets is performed without the need of microstructures, leading to increased scalability and flexibility compared with continuous-flow biochips [13]. This generation is also referred to as "digital microfluidics", due to the analogy between the droplets and the bits in a digital system. Such biochips, consisting of hundreds [1] and thousands [15] of cells have already been successfuly designed and commercialized. In this paper, we are interested in the second generation, droplet-based digital biochips.

1.1 Related work

Researchers have initially addressed separately architectural and physical-level synthesis of DMBs. Su and Chakrabarty [16] have proposed an integer linear programming (ILP) model for scheduling and binding, considering a given allocation, but without addressing placement and routing. During the physical-level synthesis, the placement [18, 25] of each module on the microfluidic array and the droplets routes [6, 23, 28] have to be determined.

A unified high-level synthesis and module placement methodology has been proposed in [17], where the focus has been on deriving an implementation that can tolerate faulty cells in the biochip array. Their algorithm was modified in [22] to include droplet-routing-aware physical design decisions. Yuh et al. [25] have proposed a synthesis and placement algorithm which uses a tree-based topological representation and is able to improve on the results from [17]. The algorithm has later been extended to consider defective cells on the biochip array [26]. In [11] we have proposed an



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

Figure 1: Biochip architecture

ILP-based architectural-level synthesis and placement approach for DMBs, and shown that considering the placement during the synthesis can significantly reduce the biochemical application completion time. We have also used ILP to generate implementations which have high chances of successful reconfiguration in case of faults [10]. Although ILP has the advantage that it produces the optimal solution, it is only feasible for limited problem sizes.

The combined architectural- and physical-synthesis problem has some similarities with the simultaneous scheduling and placement problem of dynamically reconfigurable field-programmable gate arrays (DR-FPGAs) [2], which is typically formulated as a 3D packing problem that minimizes the volume, seen as area $\times$ execution time. Bazargan et al. [2] have proposed offline algorithms for statically reconfigurable FPGAs and online algorithms for dynamically reconfigurable FPGAs. Yuh et al. [27] use a 3D transitive closure subGraph for the 3D packing problem. Their earlier work on temporal floorplanning using a tree-based topological representation [24] has been extended for DMBs [25].

However, there are three main differences when doing scheduling and placement for DMBs: (1) the virtual devices (created by grouping adjacent cells) can easily be moved *during the execution of operations* without incurring a significant overhead—see Section 2.1 for details; (2) non-reconfigurable devices, such as reservoirs and detectors also have to be considered; and (3) additional operations have to be introduced to temporarily store a droplet in-between operations that are not scheduled at consecutive time-steps.

1.2 Contribution

In this paper, we propose a Tabu Search-based synthesis approach that, 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 with module placement. All of previous approaches to DMBs and DR-FPGAs consider the placement of a module fixed throughout its operation. However, our scheduling and placement steps consider moving the virtual devices *during their operation* to improve the biochemical application completion time on the given biochip area. We show that by taking advantage of the dynamically reconfigurable characteristic of DMBs, significant improvements can be obtained in the application completion time, allowing us to use smaller area biochips and thus reduce costs.

The paper 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 2.3 and illustrate the design tasks using several examples. The proposed approach is presented in Section 3 and evaluated in Section 4. The last section presents our conclusions.

2. SYSTEM MODEL

2.1 *Biochip Architecture*

In a digital microfluidic biochip the manipulation of liquids is performed using discrete droplets. There are several mechanisms for droplet manipulation [8]. Our work considers electrowetting-on-dielectric (EWD) [13], but can be extended to handle other techniques as well. EWD is the most promising technique, and can provide high droplet speeds of up to 20 cm/s [13]. A biochip is composed of several cells, see Fig. 1b. The schematic of a cell is presented in Fig. 1a. The droplet is sandwiched between two glass plates (the top plate and the bottom plate), and moves within a filler fluid. The top plate contains a single ground electrode, while the bottom plate has several control electrodes. The electrodes are insulated from the droplet trough an insulation material. With EWD, the movement of droplets is controlled by applying voltages to the required electrodes. For example, turning off the middle control electrode and turning on the right control electrode in Fig. 1a will force the droplet to move to the right. For the details on EWD, the reader is directed to [13].

Several cells are put together to form a two-dimensional array (an example architecture is presented in Fig. 1b). Using EWD manipulation, droplets can be moved to any location without the need for pumps and valves, which are required in a continuous-flow biochip. Besides the basic cell discussed previously, a chip typically contains input and output ports and detectors. The detection can be done by using, for example, a light-emitting diode (LED) beneath the bottom plate and a photodiode on the top plate. The chip shown in Fig. 1b can be used for the diagnosis of metabolic disorders, by measuring the lactate and glucose level in human physiological fluids. Hence, the device contains the necessary input ports for introducing the samples (urine, plasma and serum) and the reagents (lactate and glutamate oxidase and buffer substance NaOH) on the microfluidic array, where the corresponding protocol will be performed. Using this architecture, and changing correspondingly the control voltages, all of the required operations, such as transport, splitting, merging, dispensing, mixing, and detection, can be performed. For example, a mixing operation on a 2×4 mixing module is shown in Fig. 1c. Mixing is done by transporting two

Table 1: Module library

Operation	Area (cells)	Time (s)
Mixing	2×4	3
Mixing	1×3	5
Mixing	2×2	10
Dilution	2×5	3
Dilution	1×3	8
Dilution	2×2	13

droplets to the same location, and then moving them next to each other according to a certain pattern, such as the one in Fig. 1c. A mixing module can be created by grouping adjacent electrodes on which the droplet will be moved. Any cells in the chip can be used for such a purpose, thus, we say that the chip is "reconfigurable".

Because of the virtual character of the mixer, the device can be shifted to a different position, by routing the droplet to another group of electrodes, where the mixing operation can be continued. Fig. 1c shows how the 2×4 mixing module can be reconfigured to occupy a different group of cells. We consider that while the mixing operation is being performed, with the droplet being on the cell denoted by C_1, we decide to change the position of the mixer. In our example, the droplet will be routed to the nearest position belonging to the new group of cells, C_2, where it will continue the initial mixing pattern. The only overhead that must be considered while moving the module is the additional time required to transport the droplet between the two positions. In this paper we consider the data[1] from [13], which allows us to approximate that the time required to route the droplet one cell is 0.01 s, which is an order of magnitude smaller than operation times, see Table 1. The routing overhead will be considered during the synthesis process.

A given pattern will require a particular chip area and operation completion time. We consider that designers will build and characterize a module library $\mathcal{L}$, where for each operation there are several options with varying areas and execution times, see Table 1.

2.2 Biochemical Application Model

We model a biochemical application using an abstract model consisting of a sequencing graph [5]. 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 set 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 Fig. 2 we have an example of an application graph with twelve operations, O_1 to O_{12}. The application consists of four mixing operations (O_7, O_8, O_9 and O_{10}), one diluting operation (O_1) and seven input operations (O_2, O_3, O_4, O_5, O_6, O_{11}, O_{12}). O_1 is a diluting operation that has two outgoing edges, representing an output of two droplets. This requires a split operation. Considering Fig. 1a, a droplet is split by turning on the left and right electrodes and turning off the middle electrode [14]. 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 the operation O_7 is bound to a 2×2 mixing module denoted by $Mixer_1$ (i.e., $\mathcal{B}(O_7) = Mixer_1$). Then, according to Table 1, the execution time for O_7 will be $C_7^{Mixer_1}$ = 10 s.

[1]Electrode pitch size = 1.5 mm, gap spacing = 0.3 mm, average linear velocity = 20 cm/s.

2.3 Problem Formulation

The problem we are addressing in this paper can be formulated as follows. Given (1) a biochemical application modeled as a graph $\mathcal{G}$, (2) a biochip consisting of a two-dimensional m×n array $\mathcal{C}$ of cells and (3) a characterized module library $\mathcal{L}$, we are interested to synthesize that implementation Ψ, which minimizes the completion time $\delta_{\mathcal{G}}$ (i.e., finishing time of the sink node, t_{sink}^{finish}).

Synthesizing an implementation $\Psi = < \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P} >$ means deciding on: (1) the 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} of each operation O_i on its corresponding module and (4) the placement $\mathcal{P}$ of the modules on the $m \times n$ array.

The next subsections will illustrate each of these tasks. The presentation order does not correspond to the order in which our synthesis approach performs these tasks. The proposed synthesis approach can be extended to take into account faults in the cells of the array, as we have shown in [11].

2.4 Allocation and placement

Let us consider the graph shown in Fig. 2. We would like to implement the operations on the 8×8 biochip from Fig. 1b. We consider the current moment of time as being t. We assume that a diluting operation from another application has been scheduled at an earlier time step on the module denoted with D_1, has been placed on the microfluidic array as shown in Fig. 3b and will finish executing at t+5. The input operations are already assigned to the corresponding input ports. Thus, O_2 is assigned to the input port S_1, O_3 to R_1, O_4 to S_2, O_5 to R_2, O_6 to S_3, O_{11} to B and O_{12} to S_2. However, for the mixing operations (O_7, O_8, O_9 and O_{10}) and the dilution operation (O_1) 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 1, without the 2×4 mixing module. We have to select modules from the library while trying to minimize the application completion time and place them on the 8×8 chip. A solution to the problem is presented in Fig. 3b–(e), where the following

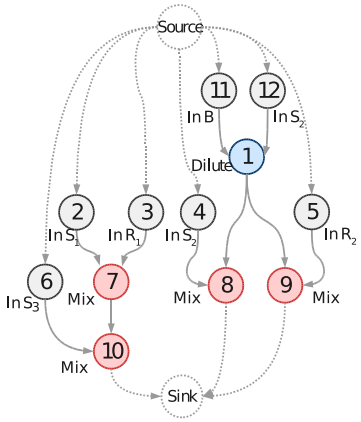
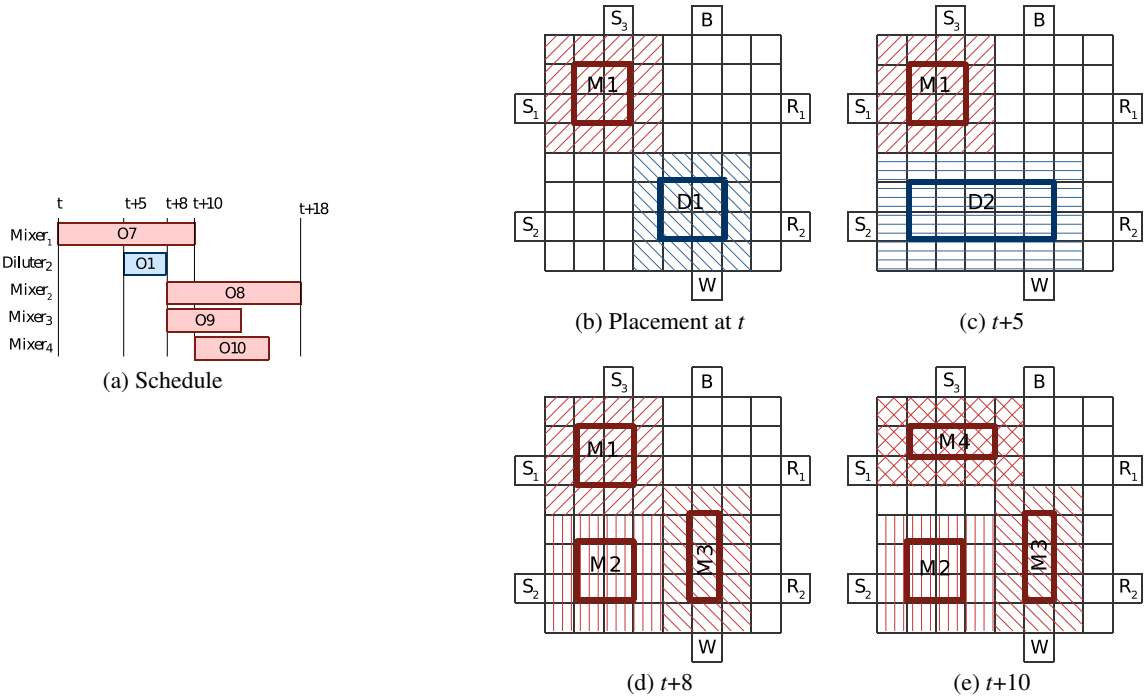


Figure 2: Application graph



(a) Schedule

(b) Placement at t

(c) t+5

(d) t+8

(e) t+10

Figure 3: Implementation example

modules[2] are used: two 2×2 mixers ($Mixer_1$, $Mixer_2$), two 1×3 mixers ($Mixer_3$, $Mixer_4$) and one 2×5 diluter ($Diluter_2$).

The placement for the solution is as indicated in Fig. 3b–(e), where we can notice that modules occupy a space larger than their size (the hashed area corresponding to each module).This is to avoid droplet-merging and contamination. If two droplets are next to each other on two adjacent cells, they will tend to merge to form one single droplet. Two approaches have been considered for solving this problem. The first approach, used in [17, 23], consists in having a one-cell distance between any two adjacent modules, which is sufficient for isolating the functional regions on which operations are executing. However, if routing is not considered at the same time with placement, the resulted solution will require significant modifications for accomodating the necessary routes. As routing needs a 3-cell width channel, one cell between adjacent modules will not be sufficient for creating the necessary routes and thus transporting the droplet. Moreover, the lack of segregation cells between reservoirs and modules placed in their proximity can make the dispensing process impossible. In the second approach, proposed in [4], a segregation area is wrapped around each module. This approach is depicted in Fig. 3b, where $Diluter_1$, which has a size of 2×2 occupies 4×4 cells. As it can be seen, module wrapping provides a 2-cell width channel between any two adjacent modules as well as a 1-cell channel between modules placed at the chip boundary and reservoirs. The advantage of this approach is that the segregation areas can be adjusted during a post-processing step to introduce the necessary paths for droplet movement. In this article, we consider the second approach, and we assume that the routing will be performed in a separate phase, after the positions of the modules have been determined.

Our placement problem has similarities with the placement of DR-FPGAs, where 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 main difference to DR-FPGAs is that we can easily move operations *during their execution*, as discussed in Section 2.1. This property will be used to improve the scheduling, see Section 2.6.

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

2.5 Binding and Scheduling

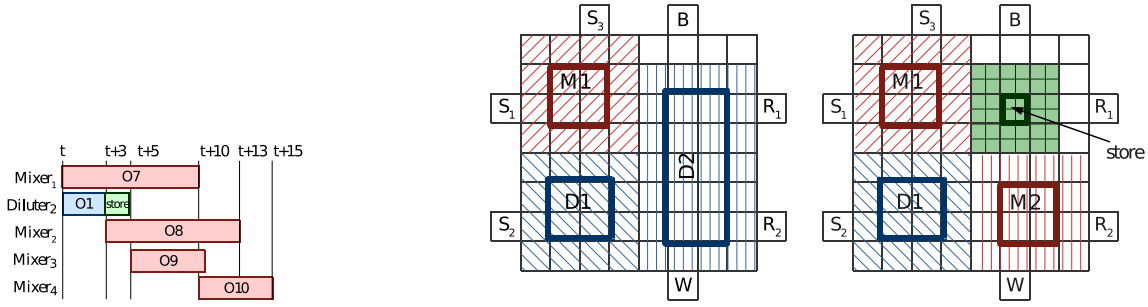
Once the modules have been allocated and placed on the cell array, we have to decide on which modules to execute the operations (binding) and in which order (scheduling), such that the application completion time is minimized.

Considering the graph in Fig. 2 with the allocation presented in the previous section, Fig. 3a presents the optimal schedule in the case of static virtual modules, whose placement remains the same throughout their operation. The schedule is depicted as a Gantt chart, where, for each module, we represent the operations as rectangles with their length corresponding to the duration of that operation on the module. For example, operation O_{10} is bound to module $Mixer_4$, starts immediately after the mixing operation O_7 (i.e, $t_{10}^{start} = t + 10$) and takes 5 s, finishing at time t_{10}^{finish} = t+15 s.

The diluting operation O_1 cannot start on module $Diluter_2$ until the operation bound to $Diluter_1$ has finished executing, at time t+5. Scheduling also decides the access to non-reconfigurable modules, such as input/output ports and detectors, but in this example we have omitted it for simplicity.

2.6 Dynamic Reconfiguration

Although the schedule presented in Fig. 3a is optimal for the given allocation and binding, it can be further improved by taking advantage of the property of dynamic reconfiguration of the digital biochip. Consider the placement in Fig. 3b. Even though the number of free cells on the microfluidic array at time t is higher than the number of cells in $Diluter_2$, the fragmentation of the space makes the placement of $Diluter_2$ impossible. Hence, the operation has to



(a) Schedule (b) Placement at t (c) Dynamic reconfigurability

Figure 4: Motivational example

wait until t+5, in Fig. 3c, when $Diluter_1$ finishes executing, and there are enough free adjacent cells for accomodating $Diluter_2$.

However, this delay can be avoided by "shifting" $Diluter_1$ to another location such that the space fragmentation is minimized. For example, by moving the module three cells to the left as in Fig. 4b, we can place $Diluter_2$ at time t, obtaining the schedule in Fig. 4a. Shifting is done by changing the activation sequence of the electrodes, such that the droplet is routed to the new position, where it continues moving according to the mixing pattern. The moving overhead is equal to the routing time to the new destination, which, under the assumptions in Section 2.1 is 3×0.01 s.

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 allocate a storage cell for $e_{i,j}$. Hence, the allocation of storage cells depends on how the schedule is constructed. In Fig. 4a one of the two droplets resulted from the diluting operation O_1 has to be stored until O_9 is scheduled for execution. The placement in Fig. 4c shows the 1×1 storage module.

3. TABU SEARCH BASED SYNTHESIS

The problem presented in the previous section is NP-complete (scheduling in even simpler contexts is NP-complete [21]). Hence, in [11] we have proposed an ILP-based method for the problem described in Section 2.3, but without considering dynamic reconfiguration. However, as biochips are becoming larger, ILP is no longer feasible for their synthesis. In addition, significant gains can be obtained by considering dynamic reconfigurability during synthesis. Our synthesis strategy, presented in Fig. 5, 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 $\Psi = <\mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{P}>$ consisting of the allocation, binding, scheduling and placement, which minimizes the schedule length $\delta_{\mathcal{G}}$ on the given biochip $\mathcal{C}$. In this paper, we use a Tabu Search (TS) metaheuristic [9] to decide the allocation $\mathcal{A}$ and binding $\mathcal{B}$ (line 3 in Fig. 5).

DMBSynthesis($\mathcal{G}$, $\mathcal{C}$, $\mathcal{L}$)

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

Figure 5: Synthesis algorithm for DMBs

For a given allocation and binding decided by TS, we use a List Scheduling (LS) heuristic [12] to decide the schedule $\mathcal{S}$ of the operations. LS is based on a sorted priority list, L_{ready}, containing the operations $O_i \in \mathcal{V}$ which are ready to be scheduled (all the predecessor operations have finished executing). During each iteration, the operation O_i with the highest priority is selected to be scheduled. The priorities Π of the operations are also decided by TS. Before a ready operation O_i can be scheduled, its corresponding module, $M_i = \mathcal{B}(O_i)$, is placed on the microfluidic array. The combined scheduling and placement is implemented by the *ScheduleAndPlace* function (line 4 in Fig. 5), which calls the *DynamicPlacement* function from Fig. 8. Once the placement is known, LS takes into account the routing time, in terms of Manhattan distance, between M_i and the source modules. The routing times, one order of magnitude smaller than operation times, were considered part of the operation times during the experiments.

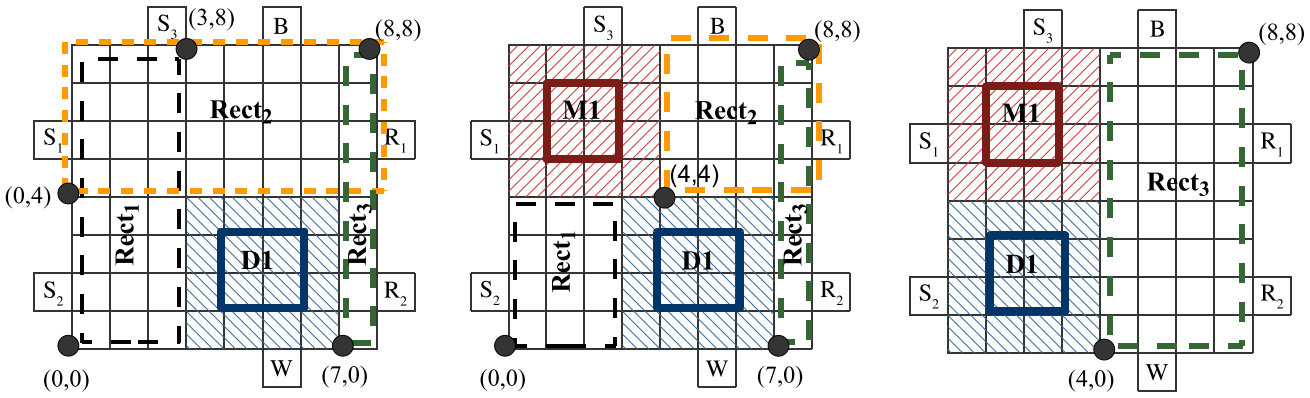
TS uses design transformations to search the solution space. Inside TS, we use the *ScheduleAndPlace* function to determine the schedule length $\delta_{\mathcal{G}}$ of each solution. TS starts from an initial solution, where each operation $O_i \in \mathcal{V}$ is bound to a randomly chosen module $M_i \in \mathcal{L}$ (line 1). The initial execution priorities are given according to the critical path priority function (line 2) [12].

The next section presents our proposed placement algorithm and Section 3.2 presents our TS implementation.

3.1 Dynamic Placement Algorithm

We have extended the online placement algorithm from Bazargan et al. [2] for DR-FPGAs to handle DMBs, where we allow dynamic reconfiguration of modules during their execution. Although the algorithm was proposed for online placement, we can use it offline, since we know beforehand all the operations that have to be executed. The algorithm from [2] has two parts: (i) a free space manager which divides the free space on the biochip into a list of overlapping rectangles, L_{rect}, and (ii) a search engine which selects an empty rectangle from L_{rect} that best accommodates the module M_i to be placed, according to a given criteria, such as "best fit". Each rectangle can be represented by the coordinates of its left bottom and right upper corners, (x_l, y_l, x_r, y_r). Our proposed algorithm, *DynamicPlacement*, is presented in Fig. 8. The placement algorithm takes as input the $m \times n$ matrix $\mathcal{C}$ of cells, the current placement of modules $\mathcal{P}$, the free space L_{rect} and the module M_i to be placed and returns the rectangle $R_i = (x_l, y_l, x_r, y_r)$ with the location of M_i. If no rectangle is found, LS will have to delay the operation corresponding to M_i.

Let us illustrate the placement algorithm by using Fig. 3b. The ready list consists of the operations in the graph that are ready to be scheduled, hence $L_{ready} = \{O_7, O_1\}$. Let us assume that O_7 has



(a) Initial placement (b) Placement at t (c) Dynamic reconfigurability

Figure 6: Dynamic placement example

DynamicPlacement($\mathcal{C}$, $\mathcal{P}$, L_{rect}, M_i)

```
1  // search for R_i ∈ L_rect that best fits M_i
2  R_i = SelectRectangle(L_rect, M_i)
3  if ∃ R_i then
4     UpdatePlacement(P, R_i)
5     UpdateFreeSpace(L_rect, R_i)
6     return R_i
7  end if
8  // dynamically reconfigure already placed modules
9  while ∄ R_i ∧ RoutingOverhead ≤ 1 s do
10    BestMove = SelectBestMove(C, P, L_rect)
11    PerformMove(BestMove, P, L_rect)
12    RecordMove(MovesList, BestMove)
13    // search for R_i that best fits M_i
14    R_i = SelectRectangle(L_rect, M_i)
15    if ∃ R_i then
16       UpdatePlacement(P, R_i)
17       UpdateFreeSpace(L_rect, R_i)
18       return R_i
19    end if
20 end while
21 // no placement has been found, restore the original P
22 UndoMoves(P, L_rect, MovesList)
```

Figure 8: Placement algorithm for DMBs

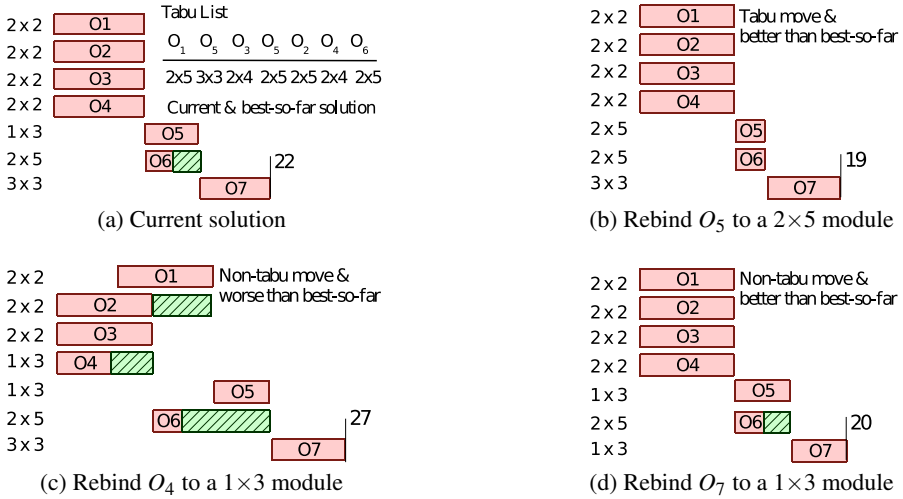
the highest priority and is bound to a 2×2 module, $Mixer_1$. The LS algorithm will select O_7 and will call *DynamicPlacement* to place $Mixer_1$ on the biochip array. The module $Diluter_1$, which is currently executing at time t, divides the free space into three overlapping rectangles $L_{rect} = \{Rect_1 = (0,0,3,8), Rect_2 = (0,4,8,8), Rect_3 = (7,0,8,8)\}$, see Fig. 6a. As rectangle $Rect_2 = (0,4,8,8)$ is the only one sufficiently large to accommodate the module, $Mixer_1$ will be placed at its bottom corner (line 2 in Fig. 8). Consequently the free space will be updated (line 5) to $L_{rect} = \{Rect_1 = (0,0,3,4), Rect_2 = (4,4,8,8), Rect_3 = (7,0,8,8)\}$ as depicted in Fig. 6b.

After the scheduling and placement of O_7, the next operation to be considered for scheduling at time t is O_1. Because of space fragmentation, no free rectangle can accommodate the 2×5 diluter currently assigned to O_1 and the operation would have to be delayed until t+5, as depicted in Fig. 3c, where the diluter is denoted with D_2. However, when no suitable rectangle can be found for accommodating a device, our algorithm, as opposed to [2], will try to decrease the space fragmentation on the microfluidic array by moving the modules during their operation.

We use a greedy approach to decide on which modules to move (line 9–20), until there is space for the current module M_i or a termination criteria is reached. As moving a device requires routing the droplet from the initial position to another one on the array, we place a constraint on the increase in routing time due to moving a device, of one time step, i.e., one second. For example, for a routing time of 0.01 s accross one cell, we move modules to accommodate the current module M_i such that routing would not increase with more than 100 cells. The routing distance is calculated based on the Manhattan distance between the left top corners of the old position and the new position of the module considered for moving. If not enough free space is thus created for M_i, we restore the previous placement (line 22). In each iteration (lines 9–20), our greedy approach selects the best move (line 10), the one which brings two rectactangles as close as possible (minimizing the Manhattan distance between the upper left corners) and at the same time increases the number of free adjacent cells that would be obtained by merging them. For example, in Fig. 3b we consider all the possible moves that can be performed on the currently placed modules, $Mixer_1$ and $Diluter_1 \in \mathcal{P}$. As we can see in Fig. 6b, $Mixer_1$ can be moved at most four cells to the right, while $Diluter_1$ can be moved at most three cells to the left and one to the right. The algorithm will choose to shift $Diluter_1$ three cells to the left, which is the best move: after the move, the Manhattan distance between $Rect_1$ and $Rect_2$ is 4 and the two rectangles contain 28 cells, corresponding to a cost of 32. The existing free space is thus merged into only one rectangle with coordinates $Rect_3 = (4,0,8,8)$. As there are now enough adjacent cells, $Diluter_1$ will be placed on the microfluidic array and the placement algorithm will terminate.

3.2 Tabu Search

Tabu Search (TS) is a metaheuristic based on a neighborhood search technique which uses design transformations (moves) applied to the current solution, $x_{current}$, to generate a set of neighboring solutions, $N_{current}$, that can be further explored by the algorithm. Our TS implementation performs two types of transformations: (i) re-binding moves and (ii) priority swapping moves. A re-binding move consists in the re-binding of a randomly chosen operation, O_i, currently executing on module M_i, to another module M_j. Such a move will take care of the allocation, e.g., removing M_i and allocating M_j. A priority swapping move consists in swapping the priorities of two randomly chosen operations in the graph.



(a) Current solution

(b) Rebind O_5 to a 2×5 module

(c) Rebind O_4 to a 1×3 module

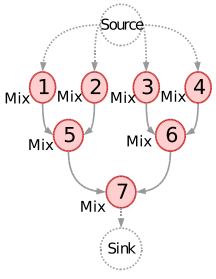
(d) Rebind O_7 to a 1×3 module

Figure 7: Tabu Search neighborhood

In order to efficiently perform the search, TS uses memory structures, maintaining a history of the recent visited solutions (a "tabu" list). By labeling the entries in the list as tabu (i.e., forbidden), the algorithm limits the possibility of duplicating a previous neighborhood upon revisiting a solution. We use two tabu lists, one for each type of move. These are constructed as attribute-based memory structures, containing not the complete recent solutions, but only relevant modified attributes, thus reducing the amount of memory required to memorize the history of the search algorithm. Hence, if an operation O_i is re-bound to a module M_j as result of a re-binding move, the change of the solution will be recorded in the corresponding tabu list as a pair of the form (O_i, M_j) and if the priorities of two operations O_i and O_j are swapped as part of the diversification process, the move will be recorded as (O_i, O_j).

However, in order not to prohibit attractive moves, an "aspiration criteria" may be used, allowing tabu moves that result in solutions better than the currently best known one. Moreover, in order to avoid getting stuck in a local optima, TS uses "diversification". This involves incorporating new elements that were not previously included in the solution, in order to diversify the search space and force the algorithm to look in unexplored areas. Based on experiments, we have decided to use priority swapping as a diversification move, only when the best known solution does not improve for a defined number of iterations, no_{div}, determined experimentally.

Let us use the mixing stage of the polymerase chain reaction (see Section 4 for details), shown in Fig. 9a, and the module library in Fig. 9b to illustrate how TS works. Consider the current solution as being the one represented by the schedule in Fig. 7a. The current tabu list, presented to the right, contains the recently performed transformations. As all operations are mixing operations, we will denote a module by its area, e.g. O_1 is bound to a mixing module of 2×5 cells. Starting from this solution, TS uses rebinding moves to generate the neighbor solutions. Out of the possible neighboring solutions we present three in Fig. 7b–(d). The solution in Fig. 7b is tabu and the one in Fig. 7c is worse than the current solution (which is the best so far). In the solution in Fig. 7d O_7 is re-bound to a new, 1×3 mixer, which results in a non-tabu solution better than the current one. However, TS will select the move in Fig. 7b, that would change the 1×3 mixer in Fig. 7a for O_5 to a 2×5 mixer module. Although the move $(O_5, 2\times 5)$ is marked as being tabu, it leads to a better result than the currently best known one and thus, according to the aspiration criteria, it is accepted. The new solution is evaluated by using the unified scheduling and placement algorithm presented before which determines the completion time $\delta_{\mathcal{G}}$ of the application graph $\mathcal{G}$. The algorithm terminates when a given time-limit for performing the search has been reached.



(a) PCR/M graph

Operation	Area (cells)	Time (s)
Mixing	2×5	2
Mixing	2×4	3
Mixing	1×3	5
Mixing	3×3	7
Mixing	2×2	10
Dilution	2×5	4
Dilution	2×4	5
Dilution	1×3	7
Dilution	3×3	10
Dilution	2×2	12
Detection	1×1	30

(b) Module library

Figure 9: Experimental evaluation

4. EXPERIMENTAL EVALUATION

In order to evaluate our proposed approach, we have used three-real life examples and ten synthetic benchmarks. The Tabu Search-based algorithm[3] was implemented in Java (JDK 1.6), running on SunFire v440 computers with UltraSPARC IIIi CPUs at 1,062 GHz and 8 GB of RAM. The module library used for all the experiments is shown in Fig. 9b.

In the first set of experiments we were interested to determine the quality of our TS approach. Hence, we have used the ILP implementation from [11] to determine the optimal solutions. ILP does not consider the dynamic reconfigurability of virtual devices during the execution of operations and is only able to produce results for smaller applications. In order to have a fair comparison with the ILP, we removed the possibility of moving devices during their execution from our TS approach. Table 2 presents the results obtained for this modified TS approach, denoted by TS^-, and ILP for two real-life examples: (1) *In-vitro* diagnostics on hu-

[3]Values for TS parameters determined experimentally: $no_{div} = 7$, length of the tabu list = 8.

man physiological fluids (IVD) [19], which has 15 operations[4] and (2) The mixing stage of a polymerase chain reaction application (PCR/M) [18], which is one of the most common techniques for DNA analysis and has 7 operations. The comparison is made for three area constraints.

Table 2: Comparison of ILP and TS$^-$ approaches

App.	Area	δ_{ILP}	ILP exec.time	δ_{TS^-}	TS$^-$ exec.time
PCR	10×10	8s	34min 22s	8s	1min
	10×9	9s	49min 23s	9s	1min
	8×10	9s	48min 52s	9s	1min
IVD	11×11	11s	75min 32s	11s	1min
	11×9	11s	78min 28s	11s	1min
	9×10	11s	85min 55s	11s	1min

As it can be seen, our Tabu Search-based approach is capable of obtaining the optimal solutions for all three areas within 1 minute CPU time-limit, while the ILP takes considerable longer, up to 85 minutes CPU time.

Another measure of the quality of a TS implementation is how consistently it produces good quality solutions. Hence, we used our TS-based approach for synthesizing a large real-life application implementing a colorimetric protein assay (103 operations), utilized for measuring the concentration of a protein in a solution.

Table 3 presents the results obtained by synthesizing the protein application on three progressively smaller microfluidic arrays. We present the best solution (in terms of schedule length), the average and the standard deviation obtained after 50 runs of the TS algorithm. Let us first concentrate on the results obtained for the case when we have used a time limit of 60 minutes for the TS. As we can see, the standard deviation is quite small, which indicates that TS consistently finds solutions which are very close to the best solution found over the 50 runs, which will explore differently the solution space, resulting thus in different solutions.

Moreover, the quality of the solutions does not degrade significantly if we reduce the time limit from 60 minutes to 10 minutes and 1 minute. This is important, since we envision using TS for architecture exploration, where several biochip architectures have to be quickly evaluated in the early design phases (considering not only different areas, but also different placement of non-reconfigurable resources such as reservoirs or detectors).

For the second set of experiments we were interested in the gains that can be obtained by allowing the dynamic reconfiguration of the devices during their execution. Hence, we have compared TS$^-$ with TS for the protein application, and the results are presented in Table 3. As we can see, taking into account the dynamic reconfigurability property of the biochip, significant improvements can be gained in terms of schedule length, allowing us to use smaller areas and thus reduce costs. For example, in the most constrained case, a 11×12 array, we have obtained an improvement of 7.68% in the average completion time compared with TS$^-$, for the same limit of time, 1 minute.

In a final set of experiments we have evaluated our proposed method on ten synthetic applications. Due to the lack of biochemical application benchmarks, we have generated a set of synthetic graphs using Task Graphs For Free (TGFF) [7]. The applications are composed of 10 up to 100 operations and the results in Table 4 show the best and the average completion time obtained out of 50 runs of TS and TS$^-$ using a time limit of 10 minutes.

[4]The input and detection operations were not considered.

Table 3: Results for the colorimetric protein assay

Area	Time limit	Best		Average		Standard dev.	
		TS	TS$^-$	TS	TS$^-$	TS	TS$^-$
13×13	60 min	179	182	187.58	189.88	2.68	2.90
	10 min	179	182	187.89	192.00	3.55	3.64
	1 min	185	191	195.13	199.20	4.27	4.70
12×12	60 min	183	182	189.76	190.86	3.01	3.20
	10 min	185	185	191.84	197.73	2.87	6.50
	1 min	187	193	206.80	212.62	7.74	10.97
11×12	60 min	182	184	191.48	192.50	3.63	3.78
	10 min	186	194	200.40	211.72	10.20	14.37
	1 min	204	226	232.80	252.19	11.34	15.76

For each synthetic application we have considered three areas, from $Area_1$ (largest) to $Area_3$ (smallest). The results in Table 4 confirm the conclusion from Table 3: as the area decreases, considering dynamic reconfiguration becomes more important, and leads to significant improvements. For example, for the synthetic application with 50 operations, in the most constrained case, a 9×9 array, we have obtained an improvement of 11.18% in the average completion time compared with TS$^-$.

5. CONCLUSIONS

In this paper we have presented a Tabu Search based-technique for the synthesis of digital microfluidic biochips. The proposed approach considers the unified architectural design (allocation, binding and scheduling) and physical design (placement of operations on a microfluidic array). In this work, we have considered that the virtual devices can be moved during the execution of their operation. Three real life examples as well as a set of ten synthetic applications have been used for evaluating the effectiveness of the proposed approach. We have shown that by exploiting the dynamic reconfigurability of digital microfluidic biochips significant improvements can be gained, allowing us to use smaller area biochips and thus reduce costs.

6. REFERENCES

[1] Advanced Liquid Logic. http://www.liquid-logic.com/technology.html.

[2] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design and Test of Computers*, 17(1):68–83, 2000.

[3] K. Chakrabarty and J.Zeng. *Design automation methods and tools for microfluidic-based biochips*. Springer, 2006.

[4] K. Chakrabarty and F. Su. *Digital Microfluidic Biochips: Synthesis, Testing, and Reconfiguration Techniques*. CRC Press, Boca Raton, FL, 2006.

[5] K. Chakrabarty and J. Zeng. Design automation for microfluidics-based biochips. *ACM Journal on Emerging Technologies in Computing Systems*, 1(3):186–223, 2005.

[6] M. Cho and D. Z. Pan. A high-performance droplet router for digital microfluidic biochips. In *Proceedings of International Symposium on Physical Design*, pages 200–206, 2008.

[7] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign*, pages 97–101, 1998.

[8] R. B. Fair. Digital microfluidics: is a true lab-on-a-chip possible? *Microfluidics and Nanofluidics*, 3(3):245–281, 2007.

Table 4: Results for synthetic benchmarks

Nodes	$Area_1$	$Average_1$		$Best_1$		$Area_2$	$Average_2$		$Best_2$		$Area_3$	$Average_3$		$Best_3$	
		TS	TS^-	TS	TS^-		TS	TS^-	TS	TS^-		TS	TS^-	TS	TS^-
10	9×7	21.94	24.00	20	20	7×8	22.56	25.19	19	20	8×6	29.12	32.58	27	27
20	8×7	55.06	55.16	55	55	7×7	58.53	58.61	58	58	6×7	62.07	67.33	61	67
30	10×11	52.23	56.00	39	41	10×10	55.35	60.78	41	41	9×11	59.60	66.52	46	54
40	10×11	63.82	68.58	56	58	10×10	69.48	76.50	56	58	9×10	76.92	86.37	66	67
50	10×10	107.24	117.78	103	104	8×11	123.70	132.44	111	112	9×9	127.50	143.56	109	119
60	11×12	111.24	113.50	107	110	10×11	112.38	115.40	109	112	9×10	117.94	125.58	112	118
70	12×12	127.69	131.87	121	121	11×12	129.72	137.66	122	123	10×11	143.44	159.72	129	136
80	12×12	159.40	161.60	151	154	11×11	186.54	192.86	165	165	10×11	196.60	210.90	165	168
90	15×15	127.64	128.02	120	120	14×14	131.96	135.68	120	127	13×13	153.86	164.20	133	142
100	15×15	175.04	178.36	163	163	14×14	175.66	179.90	161	170	13×13	175.42	183.84	170	175

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

[10] E. Maftei, P. Paul, J. Madsen, and T. Stidsen. 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*, pages 425–430, 2008.

[11] E. Maftei, P. Paul, and F. P. Vlădicescu. Synthesis of reliable digital microfluidic biochips using Monte Carlo simulation. In *Proceedings of the European Safety and Reliability Conference*, pages 2333–2341, 2008.

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

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

[14] H. Ren, V. Srinivasan, and R. B. Fair. 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*, pages 619–622, 2003.

[15] Silicon Biosystems. http://www.siliconbiosystems.com.

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

[17] F. Su and K. Chakrabarty. Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips. In *Proceedings of the 42nd annual conference on Design automation*, pages 825–830, 2005.

[18] F. Su and K. Chakrabarty. Module placement for fault-tolerant microfluidics-based biochips. *ACM Transactions on Design Automation of Electronic Systems*, 11(3):682–710, 2006.

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

[20] T. Thorsen, S. Maerkl, and S. Quake. Microfluidic largescale integration. *Sci.*, 298:580–584, 2002.

[21] D. Ullman. NP-complete scheduling problems. *Journal of Computing System Science*, 10:384–393, 1975.

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

[23] P. H. Yuh, S. Sapatnekar, C.-L. Yang, and Y.-W. Chang. A progressive-ILP based routing algorithm for cross-referencing biochips. In *Proceedings of Design Automation Conference*, pages 284–289, 2008.

[24] P. H. Yuh, C.-L. Yang, and Y.-W. Chang. Temporal floorplanning using the T-tree formulation. In *Proceedings of International Conference on Computer Aided Design*, pages 300–305, 2004.

[25] P. H. Yuh, C. L. Yang, and Y. W. Chang. Placement of digital microfluidic biochips using the T-tree formulation. In *Proceedings of Design Automation Conference*, pages 931–934, 2006.

[26] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang. Placement of defect-tolerant digital microfluidic biochips using the T-tree formulation. *ACM Journal on Emerging Technologies in Computing Systems*, 3(3), 2007.

[27] P.-H. Yuh, C.-L. Yang, Y.-W. Chang, and H.-L. Chen. Temporal floorplanning using three dimensional transitive closure subGraph. *ACM Transactions on Design Automation of Electronic Systems*, 12(4), 2007.

[28] Y. Zhao and K. Chakrabarty. Cross-contamination avoidance for droplet routing in digital microfluidic biochips. In *Proc. Des., Automat. Test Conf.*, pages 1290–1296, 2009.