Placement of Defect-Tolerant Digital Microfluidic Biochips Using the T-tree Formulation

PING-HUNG YUH, CHIA-LIN YANG, and YAO-WEN CHANG
National Taiwan University

Droplet-based microfluidic biochips have recently gained much attention and are expected to revolutionize the biological laboratory procedures. As biochips are adopted for the complex procedures in molecular biology, its complexity is expected to increase due to the need of multiple and concurrent assays on a chip. In this article, we formulate the placement problem of digital microfluidic biochips with a tree-based topological representation, called *T-tree*. To the best knowledge of the authors, this is the first work that adopts a topological representation to solve the placement problem of digital microfluidic biochips. We also consider the defect tolerant issue to avoid to use defective cells due to fabrication. Experimental results demonstrate that our approach is more efficient and effective than the previous unified synthesis and placement framework.

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

General Terms: Algorithms, Performance, Design

Additional Key Words and Phrases: Microfluidics, biochip, placement

ACM Reference Format:

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, Article 13 (November 2007), 32 pages. DOI = 10.1145/1295231.1295234 http://doi.acm.org/10.1145/1295231.1295234

This article is an extended and revised version of the paper presented at the 2006 IEEE/ACM Design Automation Conference (DAC) © ACM 2006.
This work was partially supported by the National Science Council of Taiwan under Grant Nos. NSC 95-2752-E-002-008-PAE and NSC 95-2221-E-002-374 and by the Excellent Research Projects of National Taiwan University, 95R0062-AE00-07.
Authors' address: P.-H. Yuh, C.-L. Yang (contact author), Department of CSIE, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan; email: {r91089, yangc}@csie.ntu.edu.tw; Y.-W. Chang, Graduate Institute of Electronics Engineering and Department of EE, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan; email: ywchang@cc.ee.ntu.edu.tw.

1. INTRODUCTION

Due to the advances in the microfabrication and microelectromechanical systems, microfluidic technology has gained much attention recently. The composite microsystems could perform conventional biological laboratory procedures on a small and integrated system. As a result, microfluidic biochips are used in several common procedures in molecular biology, such as the clinic diagnosis and the DNA sequence analysis.

Most recently, second-generation (digital) microfluidic biochips, which are based on the manipulation of the discrete liquid particles (the *droplets*), have been proposed [tutorgig]. Each droplet can be independently controlled by the electrohydrodynamic forces generated by an electric field. The field can be generated by an individually accessible electrode. Compared with the first-generation microfluidic biochips that are based on the continuous fluid flow and contain external pressure sources (e.g., micropumps), the droplets can be moved anywhere in a 2D array to perform the desired chemical reaction and the electrodes can be reprogrammed for different bioassays. With these two properties, digital microfluidic biochips can handle large-scale and complex procedure, since the complex procedure can be built based on a set of fundamental operations, such as droplet generation, mixing of multiple droplets, droplet transportation, droplet dilution, and droplet fission. Moreover, digital microfluidic biochips can be reconfigured for different levels of hierarchy.

As biochips are adopted for complex procedures in molecular biology, the design complexity of digital microfluidic biochips is expected to increase due to the need of multiple and concurrent assays on a biochip. The International Technology Roadmap for Semiconductors (ITRS) clearly points out that the integration of electro-biological devices is one of the major challenges of system integration beyond 2009 [ITRS]. Besides, the increase in the density of assays and area of digital microfluidic biochips may reduce yield. Since we need time to ramp up the yield of biochips, it is desirable to perform a bioassay on a biochip with the existence of defects. How to incorporate the defect tolerant issue in Computer Aided Design (CAD) support becomes an important issue.

Figure 1 shows the schematic view of a digital microfluidic biochip based on the principle of electrowetting on dielectric (EWOD) [Fair et al. 2003]. There are three major components in a biochip: 2D microfluidic array, reservoirs/dispensing ports, and optical detectors. The 2D microfluidic array consists of a set of basic cells with the same architecture. The 2D microfluidic array is used for the chemical reaction of droplets and droplets transportation. With this 2D array, fundamental microfluidic operations (e.g., mix, dilute, and store) can be performed for different bioassays. The mix operation is to mix two droplets containing analytes and reagents. The two droplets route to the same location and turn around some pivot points for fast mixing process. This operation can be used for preprocessing, sample dilution, or reaction between samples and reagents. The dilution operation is to mix samples with buffers to reduce the sample concentration. The dilution ratio is controlled by a hierarchy of binary mixing phases by mixing samples (or diluted droplets) and buffers. The storage

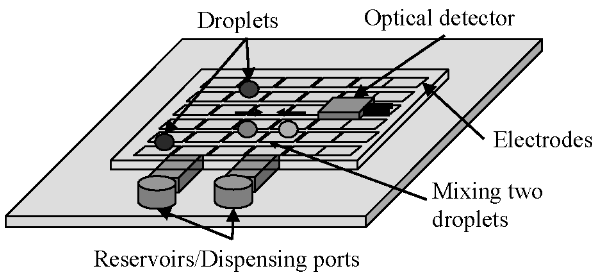


Fig. 1. The schematic view of digital microfluidic biochips.

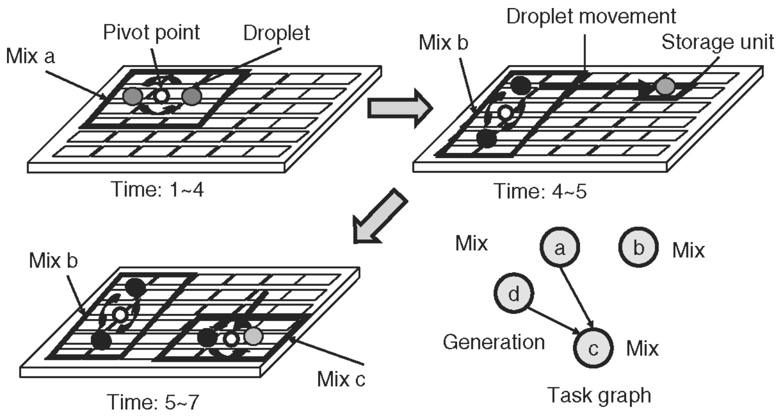


Fig. 2. An example of bioassay execution on digital microfluidic biochips.

operation is to temporarily preserve samples; that is, a droplet containing biological sample is located at a fixed location for a period of time. Note that since droplets can move freely on the 2D array, these operations can be performed anywhere in this 2D array. In other words, a basic cell can perform different operations at different time steps. This property is referred to as the *reconfigurability* of biochips. Moreover, there are different implementations of these operations with different areas and durations. For example, a mix operation can be performed in a 2×3 or 1×4 region with different mixing times. In this paper, we refer to this type of operations (mix, dilute) as the *reconfigurable operations*. The reservoirs/dispensing ports are responsible for droplets generation while the optical detectors are used for the detection of reaction results. In contrast to the 2D array, these devices have only one functionality. Therefore, in this paper, we call these device as the *non-reconfigurable devices*. The operations (e.g., droplet dispensing/generation and detection) performed on these devices are referred to as *non-reconfigurable operations*.

The bioassay is a procedure to determine the strength or activity of a biological sample by comparing its effect with those standard preparations on the living cells. Figure 2 shows the bioassay execution on a biochip. A bioassay is represented as a task graph, where each node represents a fundamental operation and each edge represents the data dependency between two operations. Each fundamental operation (mix, dilution, etc.) occupies a certain area and lasts for a period of time. There are two important characteristics of the execution of a bioassay. First, due to the reconfigurability of biochips, two fundamental operations may share the same area at different time steps. For example, the mix operation b and the mix operation a share the same area. Second, a storage unit is required to temporarily store the intermediate result between two data-dependent fundamental operations. For example, although the mix operation a is finished at time 4, the mix operation c cannot immediately start its execution since another input droplet of mix c is not available at that time. Therefore, a storage unit is required to store the result of the mix operation a. The above two characteristics complicates the placement of fundamental operations, since we need to determine not only the physical location but also the starting time of each fundamental operation. Moreover, we also need to determine the number of storages and the locations of them.

Due to the reconfigurability, the placement problem of digital microfluidic biochips includes architectural-level synthesis (i.e., scheduling and resource binding) and physical placement. How to simultaneously perform architectural-level synthesis and physical placement is the most challenging issue in the placement problem.

1.1 Related Works

Architectural-level synthesis and physical placement of digital microfluidic biochips have been addressed in the recent literature. For the architectural-level synthesis, Ding et al. [2001] proposed an architectural-level modeling and an integer linear programming based optimization method for droplet-based microfluidic biochips. Su et al. [Su and Chakrabarty 2004] used the sequencing graph to represent the bioassay protocol and proposed an integer linear programming based formulation and two heuristics, the modified list scheduling algorithm and the genetic algorithm, to solve this problem. Recently, Ricketts et al. [2006] proposed the hybrid priority scheduling algorithm based on the genetic algorithm. For the physical placement, Su et al. [Su and Chakrabarty 2005a] proposed a simulated annealing-based algorithm for the physical placement problem with given scheduled operations. They also considered the fault tolerance issue by modeling the degree of faults and identifying the empty spaces to recover operations with faulty cells. Recently, Su and Chakrabarty [2005b] presented a unified synthesis and placement flow based on parallel recombinative simulated annealing. Their algorithm consisted of three stages: binding, scheduling, and physical placement. They used the list scheduling algorithm for scheduling and a greedy placement algorithm for physical placement. They also considered the defect tolerant issue for yield enhancement.

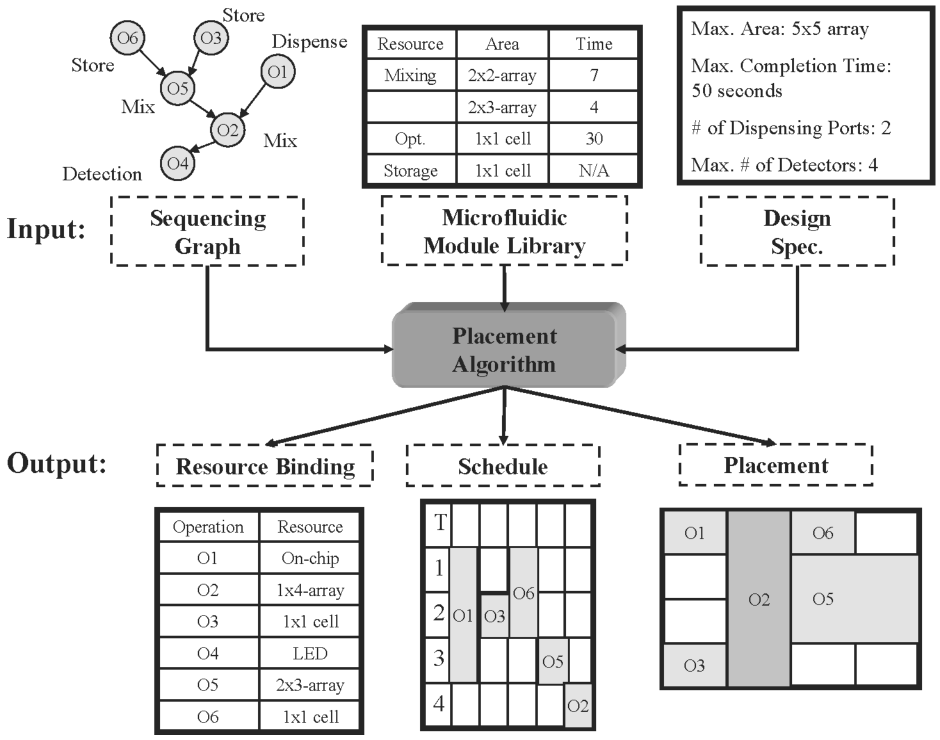
The synthesis and physical placement problem of digital microfluidic biochips are closely related to the operations of dynamically reconfigurable FPGAs (DRFPGAs), which have received much attention recently [Bazargan et al. 2000]. The digital microfluidic biochips offers the same partial reconfigurability as the DRFPGAs. Many approaches, such as the graph-theoretic approach [Fekete et al. 2001] and the topological representation based approach [Yuh et al. 2004; Yuh et al. 2004], have been proposed. Among these approaches, the T-tree [Yuh et al. 2004] representation is the state-of-the-art method for DRFPGAs.

1.2 Our Contribution

In this article, we adopt the T-tree topological representation [Yuh et al. 2004] to solve the placement problem of digital microfluidic biochips. Due to the reconfigurability of DRFPGAs, the placement of digital microfluidic biochips is similar to the simultaneous scheduling and placement of DRFPGAs. However, the placement of biochips is more complicated than that of DRFPGAs for two reasons. First, besides reconfigurable operations, biochips also have non-reconfigurable operations. Second, a storage unit is required for two data-dependent operations if they are not scheduled at consecutive time steps. To the best knowledge of the authors, our work is the first to apply a topological representation to solve the placement problem of digital microfluidic biochips. We choose the T-tree [Yuh et al. 2004] topological representation over other 3D representations, such as 3D-subTCG [Yuh et al. 2004], Sequence Triplet [Yamazaki et al. 2000], and 3D slicing tree [Cheng et al. 2005], because T-tree is effective and efficient on volume optimization and handling the storage units. We also explore the property of a bioassay to develop a clustering algorithm; since a generation operation and a reconfigurable operation are performed sequentially in a bioassay, we cluster the two operations a priori for better solution quality and less CPU time. Due to the need to perform a bioassay on a biochip with the existence of defects, the proposed placement algorithm handles the defect tolerant issue by modeling each defective cell as an obstacle and not allowing overlaps among operations and obstacles.

We evaluated the proposed placement algorithm on the colorimetric protein assay [Srinivasan et al. 2004] and the multiplexed in-vitro diagnostics [Su and Chakrabarty 2004]. We assumed different design specifications, for example, fixed architecture and limited assay completion time, for each bioassay. The experimental results show that our placement algorithm can satisfy all design specifications for all bioassay while both Su and Chakrabarty [2005b] and the 3D-subTCG based algorithm can satisfy only some of them. Moreover, we can achieve smaller volume than Su and Chakrabarty [2005b] and 3D-subTCG. For example, for the in-virto diagnostics, Su and Chakrabarty [2005b] obtains, on average, 4.07X larger volume than our algorithm. For the defect tolerance, we performed four different sets of experiments (two sets with three and two sets with four defective cells). With the existence of defects, our placement algorithm could achieve a biochip with only 16% increase in the assay completion time compared with that of a nondefective biochip with reasonable CPU time.

The remainder of this article is organized as follows. Section 2 formulates the placement problem of digital microfluidic biochips. Section 3 reviews various 3D floorplan representations and gives the advantages of T-tree over other representations. Section 4 presents the T-tree based formulation for the placement problem. Section 5 describes our temporal floorplanning algorithm. Section 6 demonstrates our defect tolerance approach. Section 7



Resource	Area	Time
Mixing	2x2-array	7
	2x3-array	4
Opt.	1x1 cell	30
Storage	1x1 cell	N/A

Max. Area: 5x5 array

Max. Completion Time: 50 seconds

# of Dispensing Ports: 2

Max. # of Detectors: 4

Operation	Resource
O1	On-chip
O2	1x4-array
O3	1x1 cell
O4	LED
O5	2x3-array
O6	1x1 cell

Fig. 3. Overview of the placement problem of biochips.

reports the experimental results. Finally, concluding remarks are given in Section 8.

2. PROBLEM FORMULATION

In this section, we give a formal definition about the placement problem of digital microfluidic biochips. Figure 3 shows the overview of the placement problem. There are three inputs to the placement problem. The first one is the sequencing graph $G = \{V, E\}$ that represents the protocol of a bioassay [Su and Chakrabarty 2004], where $V = \{v_1, v_2, \ldots, v_m\}$ represents a set of m assay operations and $E = \{(v_i, v_j), 1 \leq i, j \leq m\}$ denotes the data dependencies between two assay operations; i.e., the *precedence constraints*. We may need at most one storage unit for each edge in G to store the intermediate data between two data-dependent operations. Throughout this paper, we use operation and task interchangeably. The second one is the microfluidic module library that contains the basic modules for biochips. Each basic module is characterized by its functionality (i.e., mix, dilution, etc.) and parameters (i.e., width, height, and operation duration). The third one is the design specification, including: (1) the fixed architecture (such as 10×10 array) and limited assay completion time (such as 400 seconds) and (2) the maximum number of instances for each type of non-reconfigurable devices; that is, the *resource constraints*.

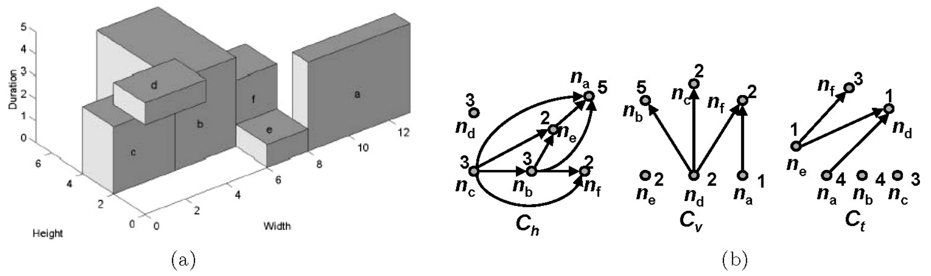


Fig. 4. (a) A 3D placement. (b) Its corresponding 3D-subTCG.

The goal of our algorithm is to simultaneously perform resource binding, scheduling, and physical placement with volume optimization under the design specification. Binding is to map an operation to a functional resource. Note that there may be several functional resources for a given operation. For example, for a reconfigurable operation, such as the mix operation, we can use a 2×2-array mixer or a 2×4-array mixer with different mixing times. However, several operations may map to the same functional resource for resource sharing. For example, we may map two detection operations to the same optical detector. After binding, the duration and dimension of each operation is determined. Scheduling is to determine the starting time of each operation under the precedence constraint. For a valid schedule, non-reconfigurable operations that map to the same device cannot execute concurrently. Physical placement is to find the physical location for each reconfigurable operation on the 2D microfluidic array. We also need to determine the locations of optical detectors. On the other hand, we can manually determine the reservoirs/dispensing ports after placement, since they do not affect the area of the biochip [Su and Chakrabarty 2005b]. In this paper, we ignore the time for transporting droplets between different tasks because the movement of droplets is very fast compared with the duration of each task [Fair et al. 2003; Srinivasan et al. 2004]. We also follow Su and Chakrabarty [2005b] in using the segregation cells to wrap each reconfigurable operation, storage unit, and optical detectors for not only providing the transportation path for droplets movement between different operations, but also isolating one operation from another to avoid unexpected cross-contamination.

3. REVIEW OF 3D FLOORPLAN REPRESENTATIONS

In this section, we briefly review popular 3D floorplan representations proposed in the recent literature, including 3D-subTCG [Yuh et al. 2004], Sequence Triplet (ST) [Yamazaki et al. 2000], T-tree [Yuh et al. 2004], and 3D slicing tree [Cheng et al. 2005]. Then we point out the advantages of T-tree over other 3D representations.

3.1 3D-subTCG

3D-subTCG is an extension of the well-known 2D floorplan representation: Transitive Closure Graph (TCG) [Lin and Chang 2001]. 3D-subTCG uses a *horizontal transitive closure graph* C_h and a *vertical transitive closure graph* C_v

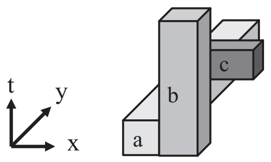


Fig. 5. A 3D floorplan; the corresponding Sequence Triplet is (*bca, bac, abc*).

to describe the geometric relations and a *temporal transitive closure graph* C_t to model the temporal relations among tasks. Figure 4 shows a 3D placement and its corresponding 3D-subTCG. Each node n_i in a transitive closure graph represents a task v_i. The value associated with a node in C_h (C_v or C_t) gives the width (height or duration) of the corresponding task, and the edge (n_i, n_j) in C_h (C_v or C_t) represents the horizontal (vertical or temporal) relation of v_i and v_j. For example, in Figure 4, since task v_c (v_a) is left to (below) v_b (v_f), there exists an edge (n_c, n_b) ((n_a, n_f)) in C_h (C_v). Similarly, since task v_a is executed before task v_d, there exists an edge (n_a, n_d) in C_t.

3.2 Sequence Triplet

Sequence Triplet (ST) is a 3D representation extended from the well-known Sequence Pair representation [Murata et al. 1995]. An ST consists of three sequences (Γ_1, Γ_2, Γ_3), where each sequence contains the label of all tasks. Figure 5 shows a simple 3D placement and its corresponding ST. ST defines the relations between two tasks based on the relative positions of this two tasks in the three sequences. The relation between two tasks is defined as follows: (1) if the sequence of two tasks v_a, v_b is the same (from left to right) in (Γ_1, Γ_2, Γ_3), i.e., (Γ_1, Γ_2, Γ_3) = (..a..b.., ..a..b.., ..a..b..), it means that task v_b is in the Y^+ direction of task v_a; (2) if (Γ_1, Γ_2, Γ_3) = (..a..b.., ..a..b.., ..b..a..), it means that task v_b is in the X^- direction of task v_a; (3) if (Γ_1, Γ_2, Γ_3) = (..a..b.., ..b..a.., ..a..b..), it means that task v_b is in the X^+ direction of task v_a; (4) if (Γ_1, Γ_2, Γ_3) = (..a..b.., ..b..a.., ..b..a..), it means that task v_b is in the Z^- direction of task v_a. For example, since the relative positions of tasks v_a and v_b satisfy relation (2), v_a is in the X^- direction of v_b.

3.3 T-tree

T-trees are a 3-ary tree, where each node corresponds to a unique task and has at most three children to represent the dimensional relationship among tasks. The T-tree is designed to represent a compacted placement where each task cannot move toward the origin. Figure 6 shows a compacted placement and its corresponding T-tree. Given a set of m tasks, let W_i, H_i, and T_i denote the width, height, and duration of each task, $1 \leq i \leq m$. We use (x_i, y_i) $((_i, y'_i))$ to denote the coordinate of the bottom-left (top-right) corner of a task v_i, and t_i (t'_i) the starting (ending) time of task v_i, $1 \leq i \leq m$. The T-tree represents the geometric relationship between two tasks as follows. If node n_j is the left child of node n_i, module v_j must be placed adjacent to module v_i in the T^+ direction, that is, $t_j = t_i + T_i$. If node n_k is the middle child of node n_i, module v_k must be placed in the Y^+ direction of module v_i, with the t-coordinate of v_k equal to

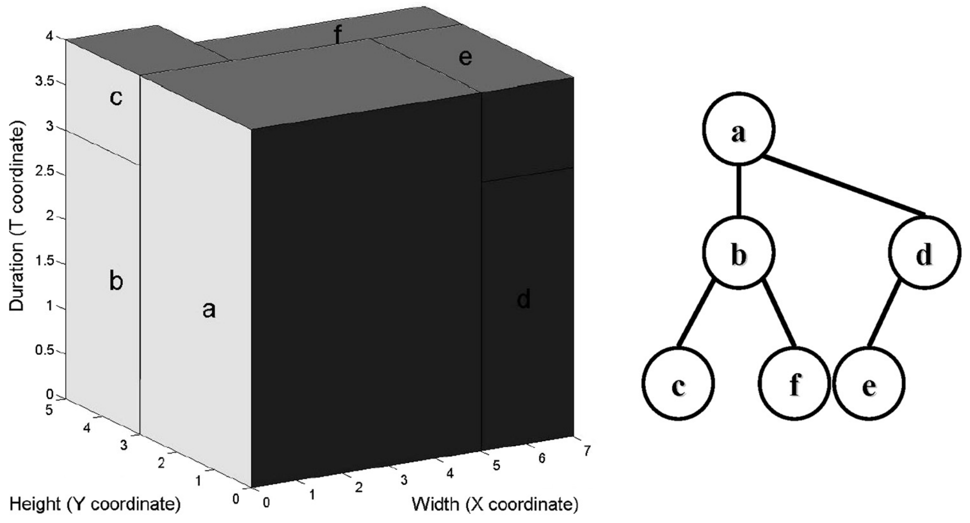


Fig. 6. A compacted placement and its corresponding T-tree.

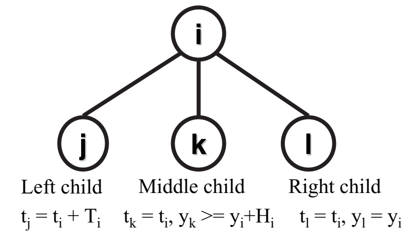


Fig. 7. The structure of the T-tree.

that of v_i, i.e., $t_k = t_i$ and $y_k \geq y_i + H_i$. If node n_l is the right child of node n_i, module v_l must be placed in the X^+ direction of module v_i, with the t- and y-coordinates equal to those of v_i, i.e., $t_l = t_i$ and $y_l = y_i$. Figure 7 shows the structure of a T-tree.

3.4 3D Slicing Tree

The 3D Slicing tree [Cheng et al. 2005] is an extension of the 2D slicing tree representation [Wong and Liu 1986]. A 3D slicing floorplan is a floorplan that consists of a finite number of nonoverlapping rectangles, where these rectangles can be obtained by repetitively subdividing rectangles with the planes that are perpendicular to the X-, Y-, or Z-axis (assume that faces of the 3D blocks are perpendicular to the X, Y, and Z axes). Figure 8(a) shows a 3D slicing floorplan. The 3D slicing tree is an oriented rooted binary tree that represents a 3D slicing floorplan. Figure 8(b) shows the corresponding 3D slicing tree of Figure 8(a). Each leaf node corresponds to a basic 3D task and is labeled by its name. Each internal node represents a supermodule and is labeled by X, Y, or Z. The label X means that the corresponding supermodule is divided into two halves by a plane that is perpendicular to the X-axis. The labels Y and Z are

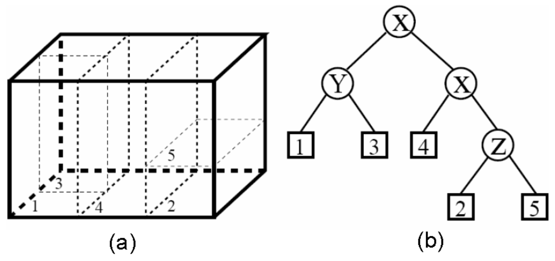


Fig. 8. (a) The 3D slicing structure. (b) The corresponding 3D slicing tree.

similarly defined. For example, the root of the two tasks v_2 and v_5 represents a supermodule. Since its label is Z, this supermodule is divided by the plane that is perpendicular to the Z-axis. Therefore, task v_5 is in the Z^+ direction of task v_2.

3.5 Discussion

In this section, we give the advantages of the T-tree over the other representations for the placement problem of digital microfluidic biochips, for which we choose the T-tree representation to solve the problem addressed in this paper:

(1) T-tree models compacted floorplans whose modules are compacted toward the origin, while the 3D-subTCG and ST model general floorplans and the 3D slicing tree models slicing floorplans. Recall that for the placement problem of digital microfluidic biochips, we need to satisfy the given fixed architecture and limited assay completion time. Therefore, a feasible 3D floorplan must be within the 3D cube defined by the fixed architecture and limited assay completion time. Since the T-tree models a compacted floorplan, it is more suitable for volume optimization, and thereby is more likely to generate solutions that are within the defined 3D cube. Consequently, T-tree is more suitable for the placement problem of digital microfluidic biochips than other 3D representations.

(2) Recall that we need a storage unit for two data-dependent operations if they are not scheduled at consecutive time steps. Also, the number and duration of these storage units are related to the schedule of operations. Based on the structure of T-tree, we can easily determine the number of storage units and their duration before packing. This process takes only linear time. As for 3D-subTCG and ST, we need, on average, $O(m^2)$ time to obtain this information with m operations since each transitive closure graph has average $\frac{m(m-1)}{6}$ edges. For the 3D slicing tree, this information cannot be obtained before packing. Further, the number of storage units varies with different schedules. We may need to delete an unused storage unit or insert a new one during simulated annealing (SA) for packing efficiency. For 3D-subTCG, when deleting an unused storage unit (inserting a new storage unit), we need to delete (insert) an edge from this storage unit to all other tasks, and detect whether the properties of 3D-subTCG are satisfied after deletion (insertion). That is, we need to detect whether the

transitive closure property is satisfied and there exists no cycles in each transitive closure graph. We need $O(|E|)$ time to detect cycles and to maintain the transitive closure property for inserting/deleting one edge, where E is the set of edges in one transitive closure graph. For ST, we need to determine the positions of a storage unit in (Γ_1, Γ_2, Γ_3) for insertion, since we need to satisfy the requirement of storage units—a storage operation v_s must be performed after the finish of v_i if v_s stores the result of v_i. This process may need $O(km)$ time, where k is the number of storage units required for insertion. For the 3D slicing tree, it is hard to guarantee that the duration of one storage unit equals the time gap between two data-dependent operations. Therefore, it is harder for a 3D slicing tree to obtain a feasible solution.

(3) As described in Section 1, the size (number of operations) of a bioassay is expected to increase because the design of a biochip will become more complicated to handle concurrent assays on a chip. Therefore, the efficiency of handling large-scale bioassays is one important factor when evaluating the suitability of a 3D representation for the placement problem of biochips. It has been shown [Yuh et al. 2004] that T-tree is more efficient and effective than 3D-subTCG and ST for the simultaneous scheduling and placement problem of DRFPGAs. Since the problem addressed here is closely related to that of the DRFPGAs, we expect the T-tree to also be more efficient and effective than 3D-subTCG and ST.

With these reasons, we decide to choose the T-tree representation to handle the placement problem of digital microfluidic biochips.

4. T-TREE BASED BIOCHIP PLACEMENT

In this section, we first present the challenges of solving the placement problem of biochips. Then we demonstrate that the execution of tasks on a biochip can be modeled as the temporal floorplanning problem. Finally, we present how to model each type of tasks with the T-tree formulation and how to handle the design specification in our algorithm.

Due to the reconfigurability of both DRFPGAs and biochips, the placement of digital microfluidic biochips is similar to the simultaneous scheduling and placement of DRFPGAs. At first glance, one may apply the techniques for DRFPGAs to solve the placement problem of biochips. However, the placement of biochips is more complicated than that of DRFPGAs based on the following two reasons.

(1) In addition to reconfigurable operations, biochips also have non-reconfigurable operations. These non-reconfigurable operations have different characteristics from that of reconfigurable operations. For example, since droplets are generated at the reservoirs/dispensing ports which are on the boundary of the 2D microfluidic array, the generation operations do not occupy the area of the 2D microfluidic array. Another example is the detection operations. Since the detectors are fixed after fabrication, two detection operations using the same detector must have the same physical

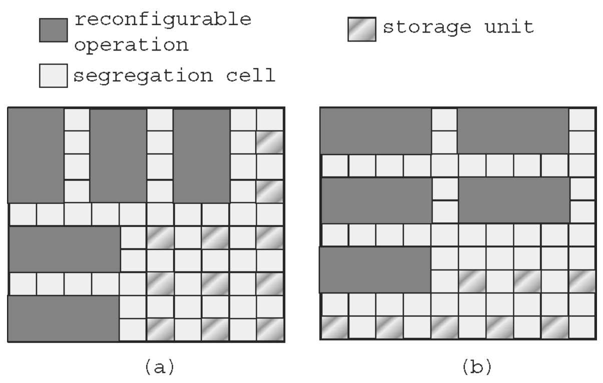


Fig. 9. The two placements with the same biochip area and number of reconfigurable operations but with different number of storage units.

location in the 2D microfluidic array. Besides, the total number of non-reconfigurable operations at any time is limited due to the limited number of non-reconfigurable devices. As a result, we require different ways to handle these non-reconfigurable operations.

(2) In digital microfluidic biochips, a storage unit is required for two data-dependent operations if they are not scheduled at consecutive time steps. The existence of the storage units complicates the placement problem for two reasons. First, instead of only area estimation as used in Su and Chakrabarty [2004], we need detail physical information of these storage units for accurate biochip area calculation. We use Figure 9 as an example to illustrate this scenario. Figure 9 shows two placements with the same biochip area (10×10) and the same number of reconfigurable operations (5 operations). Note that for the purpose of functional isolation and droplet transportation, we use the segregation cells to wrap each operation and storage unit. Although these two placements have the same biochip area, they differ in the number of storage units. If no detail physical information of the storage units is provided, we cannot obtain the exact biochip area. We may conclude that the area of Figure 9(a) is larger than that of Figure 9(b), since the number of storage units of Figure 9(a) is larger than that of Figure 9(b) if only area estimation of storage units is used.
Second, the number and duration of the storage units are not determined a priori. More importantly, the number and duration of the storage units are *related* to current schedule of operations. This characteristic makes storage units different from modules in traditional temporal floorplanning problem, where the number of modules and the volume of each module are inputs and remain unchanged during floorplanning. We use Figure 10 as an example to explain this characteristic. In this figure, we model each operation as a 3D box. Figure 10 shows a bioassay with two data-dependent operations. For each operation and storage unit, we model it as a 3D box. As shown in Figure 10(a), since operation v_b starts right after operation v_a finishes, we do not need a storage unit. Figure 10(b) shows its corresponding 3D placement with two operations. Now suppose that v_b starts one time unit after v_a

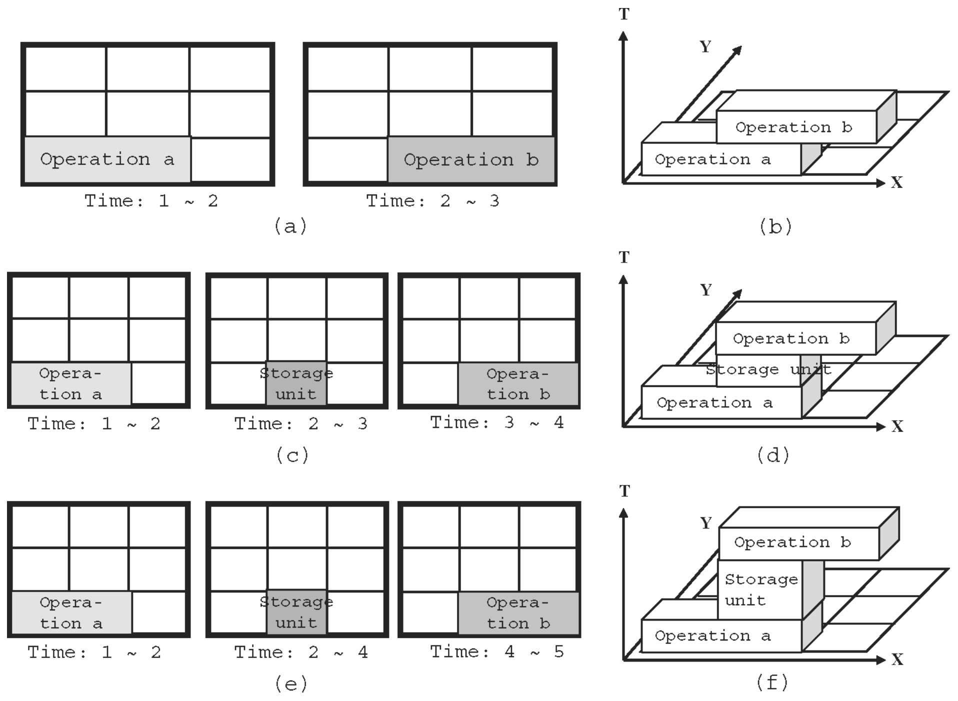


Fig. 10. Examples to show the characteristics of storage units. (a) Two tasks on a biochip. (b) The corresponding 3D floorplan of (a). (c) Two tasks with a storage unit. (d) The corresponding 3D floorplan of (c). Note that we have three modules in this floorplan. (e) Two tasks with a storage unit. (f) The corresponding 3D floorplan of (e). The duration of the storage unit is increased from 1 to 2 time units.

finishes, as shown in Figure 10(c). In this situation, we need one storage unit to store the intermediate result between v_b and v_a. Figure 10(d) shows its corresponding 3D placement. Note that compared with the 3D placement shown in Figure 10(b), we now have three modules corresponding to two operations and one storage unit v_s. Therefore, the number of storage units is related to the schedule of the data-dependent operations. Figure 10(e) shows another scenario, where v_b starts two time units after v_a finishes. Figure 10(f) shows the corresponding 3D placement. Compared with the 3D placement shown in Figure 10(d), the duration of v_s is increased from one time unit to two time units. This is because the duration of v_s must cover the time difference between v_a and v_b. As a result, the duration of storage units is also related to the schedule of two data-dependent operations. Another observation from Figure 10 is that the starting time of v_s is equal to the ending time of v_a and the ending time of v_s is the same as the starting time of v_b, as shown in Figures 10(d) and (f). We need to satisfy the above requirements of storage units when solving the placement problem.

Now we present our T-tree based placement formulation. Due to the reconfigurability of biochips, the execution of a set of tasks can be viewed as a 3D floorplan as shown in Figure 11. The X and Y dimensions give the area of a

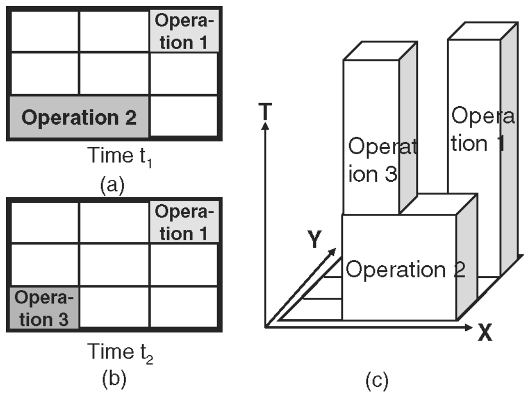


Fig. 11. (a) Two operations are executed at time $t1$. (b) At time $t2$, operation 3 starts to execute at the same physical location as operation 2. (c) The 3D modeling of the execution of the three operations.

biochip while the T dimension represents the duration of a bioassay. Suppose that both operations v_1 and v_2 are executed at time $t1$, as shown in Figure 11(a). Figure 11(b) shows that at time $t2$, we can perform operation v_3 at the same physical location as operation v_3 after operation v_2 is finished. The execution of the three operations can be modeled as a set of 3D modules with their widths and heights (X and Y dimensions) representing the physical dimensions occupied by the operations in a biochip and its duration (T dimension) being the execution time required for operations, as shown in Figure 11(c). Since the execution of a set of operations can be mapped to a 3D floorplan, we can apply the temporal floorplanning techniques to solve the placement problem of digital microfluidic biochips.

For each task in a sequencing graph, we create a unique node in a T-tree. Note that there are both reconfigurable and non-reconfigurable tasks in a biochip. For reconfigurable tasks and detection tasks, since we need to perform this type of tasks in the 2D microfluidic array, we model it as a 3D box. For non-reconfigurable tasks except the detection tasks, since the reservoirs and dispensing ports are outside the 2D microfluidic array as shown in Figure 1, we need only to consider the time aspect for this type of tasks. Therefore, we model it as a 3D line with both its width and height being zero.

In this paragraph, we describe how we model the storage units. We create a node n_s for each storage unit v_s. Since v_s holds the intermediate data between two data-dependent tasks v_i and v_j, v_s must satisfy the *storage constraint*. The storage constraint states that the starting time of v_s must be equal to the ending time of v_i and the ending time of v_s must be equal to the starting time of v_j. Figure 12 illustrates how to find the feasible locations for n_s in a T-tree to satisfy the storage constraint. Suppose that we want to find the feasible locations for n_s. Recall that if n_j is the left child of n_i, the starting time of v_j is the same as the ending time of v_i. Otherwise, the starting time of v_j is the same as the starting time of v_i. Thus, based on the structure of T-tree, the starting time of v_c in Figure 12 is the same as the ending time of v_a, and the starting times of

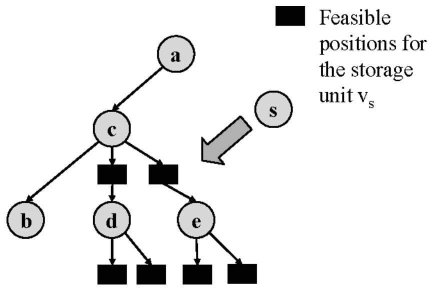


Fig. 12. An example of finding the feasible locations for a storage unit in a T-tree. Suppose that tasks v_a and v_b need a storage unit v_s.

v_b, v_c, and v_d are the same. Based on above observation, the feasible locations of n_s are the middle or the right child of nodes n_b, n_c, or n_d, as the black boxes shown in Figure 12. After placing n_s in its feasible location, we set the ending time of v_s as the starting time of v_b. Note that the duration of v_s is not fixed; it varies based on the starting time of v_b.

The design specification describes the fixed architecture, the limited assay completion time, and the resource constraints. We model the fixed architecture and limited assay completion time as the *fixed-cube constraint*. The fixed-cube constraint states that a feasible 3D floorplan must be within a 3D cube. To handle the resource constraints, we introduce the concept of the *virtual* precedence constraints. If two non-reconfigurable tasks are bound to the same non-reconfigurable resource, such as the same dispensing port, these two non-reconfigurable tasks cannot be executed at the same time. Therefore, we add an additional edge between these two tasks in the sequencing graph to satisfy the resource constraint. Note that there is no storage unit requirement in these additional edges.

5. THE FLOORPLANNING ALGORITHM

Our algorithm is based on the simulated annealing (SA) method [Kirkpatrick et al. 1983]. We adopt SA instead of genetic algorithm (GA) as our optimization method because it has been shown that SA is typically more efficient and economical than GA for the problems in electronic design automation (EDA). GA needs to maintain a set of solutions, called the population. At each iteration, GA needs to evaluate the fitness function for each solution in the current population. On the other hand, SA maintains only two solutions—the current solution and the best one. At each iteration, SA needs only to evaluate the current solution. As a result, SA typically needs less CPU time than GA. Moreover, SA uses less memory than GA due to the smaller number of solutions maintained.

Before performing SA, we first cluster one generation operation with one reconfigurable operation to reduce the CPU time and to increase the chance of obtaining more compact 3D floorplans. During SA, given a feasible T-tree, we perturb it to obtain another feasible T-tree through a set of predefined SA operations. After perturbation, we perform a feasibility detection and tree reconstruction process to obtain a feasible topology with respect to the precedence

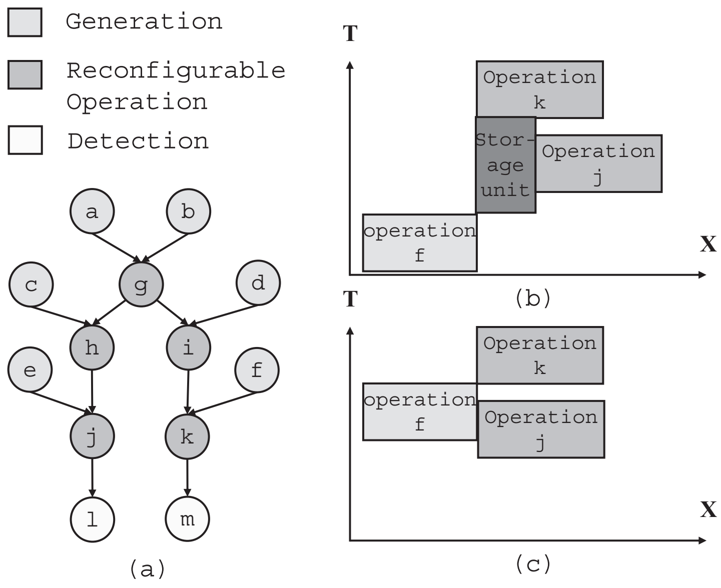


Fig. 13. (a) A sample sequencing graph. (b) A partial floorplan with v_f being scheduled long before v_k starts. (c) Another partial floorplan with v_f being scheduled right before v_k starts.

constraints and the storage constraints. Finally, a packing procedure that places all operations and optical detectors is invoked to evaluate the solution quality.

5.1 Clustering of Generation and Reconfigurable Operations

In this section, we detail our clustering algorithm. The goal of the clustering algorithm is to obtain a more compact 3D floorplan and to reduce the CPU time by reducing unnecessary storage units. This clustering algorithm is motivated by two observations. First, a generation operation and a reconfigurable operation are always performed in sequence, since we need to first generate a droplet and then to perform reactions. Second, we may improve the solution quality (e.g., volume) and reduce the CPU time by reducing the amount of storage units required via clustering. Recall that we need a storage unit for two data-dependent tasks if they are not scheduled at consecutive time steps. The duration of this storage unit also varies based on the starting and ending times of these two tasks. Since the storage units occupy certain volumes, the number and duration of them have great effect on the total volume of a 3D floorplan. If we can minimize the volume of these storage units, we may obtain a more compact 3D floorplan. We use the sample sequencing graph shown in Figure 13(a) as an example. Figure 13(b) shows a partial floorplan. For simplicity, we only show the X and T dimensions.[1] In this floorplan, since task v_f finishes much earlier than task v_k starts, the storage unit v_s will have a very long duration. Therefore, we may obtain a less compact 3D floorplan due to the non-overlapping requirement among v_s and other tasks. On the other hand, Figure 13(c) shows another

[1]For illustration purpose, in this figure, the width of the generation operation v_f is not zero.

partial floorplan. In this floorplan, since v_f finishes right before v_k starts, we do not need a storage unit between v_f and v_k. By scheduling a generation operation as near as its data-dependent reconfigurable operation, we can effectively minimize the unnecessary volume occupied by a storage unit. In this way, we have a higher chance to obtain a 3D floorplan with smaller area.

The idea of the proposed clustering algorithm is that: given a sequencing graph G, we randomly cluster one generation operation v_g with one reconfigurable operation v_r if there exist an edge between v_g and v_r in G. After clustering, the ending time of v_g is the same as the starting time of v_r. By this method, we do not need the storage unit between v_g and v_r. The other advantage is that we can reduce the number of nodes in a T-tree to speed up the packing process. However, one disadvantage is that we may potentially increase the assay completion time. The reason is as follows. Recall that we assign virtual precedence constraints among tasks that are bound to the same non-reconfigurable device. Suppose that there exists a virtual precedence constraint between two generation operations v_g and v_q. If we cluster v_g and v_r, we merge v_g and v_r into a new task v_l. So now we have a virtual precedence constraint between v_l and v_q. This means that v_q starts after v_l finishes rather than after v_g finishes. Therefore, the assay completion time is potentially increased due to clustering. In order not to increase the assay completion time, we do not actually cluster v_g and v_r into a new task. In our current implementation, we add additional requirement on nodes n_g and n_r in a T-tree. We require that n_r will always be the left child of n_g in a T-tree. This requirement has the same effect as clustering two tasks, since the ending time of v_g is the same as the starting time of v_r if n_r is the left child of n_g. In our floorplanning algorithm, if we perform SA operation on n_g, we also perform the same SA operation on n_r. We also check if the two clustered nodes are in their correct positions in a T-tree during feasibility detection and tree reconstruction process, which will be presented in Section 5.5.

5.2 Perturbation

The original SA operations defined in Yuh et al. [2004] contain Move, Swap, and Rotation. For the placement of digital microfluidic biochips, we introduce a new type of SA operations, called *Rebind*. Rebind is to bind a task to another functional resource. For a reconfigurable task, such as the mix operation, we randomly select a resource instance for this task. For example, we can change from a 2×2-array mixer to a 2×4-array mixer with different mixing times. For a non-reconfigurable task, we randomly change a task from one instance to another. For example, suppose that we have two optical detectors p_1 and p_2 for detection operations. For a detection operation v_d that originally uses p_1, we can rebind it to p_2. Note that since we add the virtual precedence constraints among tasks corresponding to the same non-reconfigurable resource instance, we modify these virtual precedence constraints after rebinding. For instance, when we bind v_d from p_1 to p_2, we delete all virtual precedence constraints of v_d and add the virtual precedence constraints between all other tasks that are bound to p_2 and v_d.

We now show how to add the virtual precedence constraints between tasks that are bound to the same non-reconfigurable device when performing the Rebind SA operation. We add the virtual precedence constraints among tasks bound to the same non-reconfigurable device based on the *execution level*, or $lv(i)$, of each operation v_i. The intuition is that if $lv(i)$ is larger than $lv(j)$, then operation v_i is executed before operation v_j. We add a virtual precedence constraint from v_i to v_j if $lv(i)$ is larger than $lv(j)$. Given a sequencing graph G, we can recursively calculate $lv(i)$ for each task v_i in G. We first assign $lv(i) = 0$ for all tasks v_i with zero out-degree in G. For example, in Figure 13(a), $lv(l)$ and $lv(m)$ are both zero. We then delete all assigned tasks and assign $lv(i) = 1$ for all remaining tasks with zero out-degree in G. For example, in Figure 13(a), $lv(j)$ and $lv(k)$ are both one. The above process repeats until all tasks are assigned its execution level.

We also enhance the original SA operations to handle the fixed-cube constraint. We bias the Move operation based on the probability of violating the fixed-cube constraint in each dimension. Let k_w (k_h, k_t) be the number of floorplans whose width (height, completion time) exceeds the user-specified width (height, completion time) in the last r iterations. In this paper, we set r equal 500. We bias the selection of the destination of the Move operation based on the values k_w/r, k_h/r, and k_t/r. For example, a larger k_w/r implies that it is more difficult to fit the floorplans to the 3D cube in the X direction. Therefore, we should try to place tasks along the Y or T directions to satisfy the fixed-cube constraint.

5.3 Placement of Optical Detectors

In this section, we describe how to place the optical detectors in our algorithm. After the chemical reaction among droplets, we need optical detectors to detect the reaction results. We need to determine the locations of these optical detectors during floorplanning. These detectors are fixed after fabrication. Therefore, if two detection operations map to the same optical detector, they should be placed at the same physical location. Note that the segregation cells are also needed for the optical detectors to avoid the optical interference.

Suppose that two detection operations v_i and v_j are bound to the same optical detector and we first determine the location of v_i. The basic idea is that we simultaneously determine the locations of v_i and v_j. Once the locations of v_i and v_j are determined, the location of the optical detector is also determined. Note that when placing the detection operations, we also warp these operations with the segregation cells. By this method, we can guarantee that the optical detectors are warped with the segregation cells after floorplanning. After determining the location of v_i, we set v_j at the same location as v_i. The original packing algorithm of T-tree maintains a list L to store all tasks whose locations are already determined [Yuh et al. 2004]. Finally, we add v_j into L to indicate that the location of v_j is already determined. Note that we need to check if v_j overlaps with any other tasks in L. If v_j overlaps with some tasks, we shift both v_i and v_j along the X direction to avoid the overlap.

5.4 Cost Function

Our goal is to simultaneously optimize the biochip area and assay completion time under the design specification. Therefore, the cost function Φ used in our algorithm is given by

$$\Phi = \alpha V/V_{norm} + \beta S/s_{norm} + \gamma M, \tag{1}$$

where V is the volume of the 3D floorplan, S is the sum of the volume of all storage units, V_{norm} is the normalized volume, S_{norm} is the normalized sum of the volumes of all storage units, and M is the penalty term for fixed-cube constraint. α, β, and γ are user-specified constants. M is defined as

$$\begin{aligned} M &= \frac{max(W_f - W_p, 0) \times W_f}{N_w^2} \\ &+ \frac{max(H_f - H_p, 0) \times H_f}{N_h^2} \\ &+ \frac{max(T_f - T_p, 0) \times T_f}{N_t^2}, \end{aligned} \tag{2}$$

where N_w (N_h, N_t) is the normalized width (height, assay completion time), W_p (H_p, T_p) and W_f (H_f, T_f) denote the width (height, assay completion time) of the design specification and a 3D floorplan, respectively. Since we must pack all modules into a pre-defined 3D cube, we penalize the excessive width, height, and completion time in the cost function. The rationale behind M is that when SA minimizes the cost function, it automatically minimizes the penalty term. Thus, we can automatically satisfy the fixed-cube constraint.

5.5 Feasibility Detection and Tree Reconstruction

After perturbation, we perform feasibility detection and tree reconstruction to satisfy all precedence constraints and storage constraints. We enhance the feasibility detection and the iterative tree reconstruction process proposed in Yuh et al. [2004] with the consideration of the storage constraints. After obtaining a feasible topology of a T-tree, we invoke the packing procedure to determine the physical locations of all tasks.

Given a T-tree H, we first check if a clustered node n_i is the left child of another clustered node n_j. If not, we Move n_i to the position of the left child of n_j. Then we check if every storage unit is in one of its feasible positions. If a storage unit n_s is not in one of its feasible positions, we Move n_s to one of its feasible positions. Note that since we modify the topology of H during the tree reconstruction process, the duration of each storage unit may change. To simplify our algorithm, we thus restrict every storage unit not to have its left child. By doing so, the starting time of a task will not be affected by any storage unit during the tree reconstruction process. Next we explain how to remove the left child of a storage unit. Suppose that a storage unit v_s stores the result of task v_a and n_k is the left child of n_s in H. We perform the move subtree procedure described below to move the subtree rooted by n_k to another place in H. First we choose one node n_z in the subtree rooted by n_a but not in

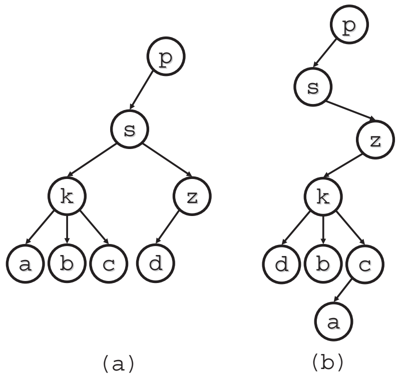


Fig. 14. The example of moving a subtree rooted by n_k as the left subtree of n_z. (a) A T-tree before moving the subtree rooted by n_k. (b) A T-tree after moving the subtree rooted by n_k as the left subtree of n_z.

the subtree rooted by n_k. Then we randomly move the subtree rooted by n_k to the positions of the left subtree, middle subtree, or right subtree of n_z based on the values of k_w/r, k_h/r, and k_t/r defined in Section 5.2. For example, if k_t/r is large, then we have lower probability to move the subtree rooted by n_k to the position of the left subtree of n_z. Without loss of generality, assume that we move the subtree rooted by n_k to the position of the left subtree of n_z. The other two cases can be handled similarly. First, if n_z has no left child, then we can simply move the subtree rooted by n_k to the position of the left subtree of n_z. Second, if n_z has its left child, we need to consider two situations:

(1) n_k has its left child: In this case, we first move the subtree rooted by n_z's left child to the position of the left subtree of n_k. Then we move the subtree originally rooted by n_k's left child to the position of the left subtree of n_f, where n_f is in the subtree rooted by n_k with no left child.
(2) n_k has no left child: In this case, we can simply move the subtree rooted by n_z's left child to the position of the left subtree of n_k.

Figure 14 gives an example if we move the subtree rooted by n_k to the position of the left subtree of n_z. Figure 15 summaries the move subtree procedure.

Once all storage units are in their feasible positions and do not have their left child, we traverse H to obtain the starting time of each task. Next, we check the precedence constraints and reconstruct H if necessary based on the method proposed in Yuh et al. [2004]. The main loop terminates when the topology of H is not changed, which means that all precedence constraints and storage constraints are satisfied. Then we assign the duration of each storage unit and adjust the number of storage units by deleting an unused storage unit and/or inserting a new one.

Note that we need to satisfy all precedence constraints after deleting or inserting a storage unit. It is easy to observe that inserting a new storage unit into one of its feasible positions does not affect the starting time of other operations. Thus, we do not violate the precedence constraints after insertion. However, deleting a storage unit with the Deletion SA operation presented

```
Subroutine: Move Subtree ($H_k$, $n_z$)
$H_k$: the subtree rooted by $n_k$;
$n_z$: the destination node;
1 Determine to make $H_k$ as the left, middle or right
2 subtree of $n_z$ based on the values $k_w/r$, $k_h/r$, and $k_t/r$;
3 if Make $H_k$ as the left subtree of $n_z$
4    if $n_z$ has no left child
5        Make $H_k$ as the left subtree of $n_z$;
6    else if $n_z$ has its left child $n_l$
7        Make $H_k$ as the left subtree of $n_z$;
8        Make $H_l$ as the left subtree of $n_k$;
9        if $n_k$ originally has its left child $n_p$
10           Make $H_p$ as the left subtree of $n_f$, where
11           $n_f$ is in $H_k$ and $n_f$ has no left child;
12 else if Make $H_k$ as the middle or right subtree of $n_z$
13       // Similar to above method;
```

Fig. 15. Summary of the Move Subtree subroutine.

in Yuh et al. [2004] may potentially violate the precedence constraints. In the following paragraphs, we present our storage adjustment process and how to modify the Deletion SA operation to satisfy the precedence constraints when deleting a storage unit.

Suppose that we want to delete a storage unit n_s. It is easy to delete n_s if n_s is a leaf or n_s has only one child. If n_s has more than one child, the original Deletion SA operation randomly chooses one of n_s's child n_c and place n_c at the original position of n_s. Then we choose one of n_c's child and place it at the original position of n_c. The process continues until a leaf node is encountered. After Deletion, the starting time of all nodes in the subtree rooted by n_c may be changed. Thus, the precedence constraints may be violated.

In this article, we modify the original Deletion SA operation when deleting n_s with two children.[2] Suppose that n_m is the middle child and n_r is the right child of n_s. When deleting n_s, instead of the node n_m itself, we place the *subtree* rooted by n_m at the original position of n_s. Then we make n_r as the right child of n_m. If n_m originally has no right child, then we are finished. Otherwise, let n_b be the original right child of n_m. We move the subtree rooted by n_b to the position of the right subtree of n_r if n_r has no right child. If n_r has its right child n_l, we find a node n_f in the subtree rooted by n_r with the same starting time as v_r and having either no middle or no right child. Then we move the subtree rooted by n_l to the position of the middle or right subtree of n_f. Figure 16 shows two T-trees before and after deleting the storage unit n_s with two children. Finally, Figures 17 and 18 summarize the storage adjustment process and the feasibility detection and tree reconstruction process, respectively.

[2]n_s has at most two children since we do not allow n_s to have its left child.

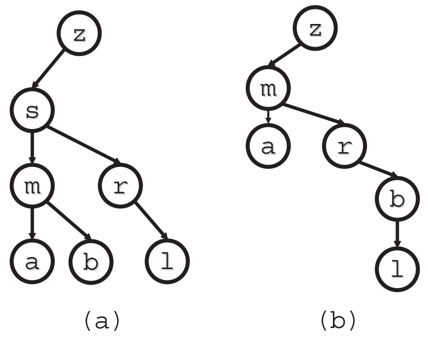


Fig. 16. The example of deleting a storage unit n_s. (a) A T-tree before deleting n_s (b) A T-tree after deleting n_s.

```
Subroutine: Storage Adjustment ($H$)
$H$: A T-tree;
1 for all storage unit $v_s$
2    if $v_s$ is required but $n_s$ is not in $H$
3        Insert $n_s$ to one of its feasible positions;
4    else if $v_s$ is not needed but $n_s$ is in $H$
5        Delete $n_s$;
6        if $n_s$ has only one child $n_o$
7            Place the subtree rooted by $n_o$ at the original
8            position of $n_s$;
9        else if $n_s$ has two children
10           Let $n_m$ ($n_r$) be the middle (right) child
11           of $n_s$;
12           Place the subtree rooted by $n_m$ at the position
13           of $n_s$;
14           Let $n_b$ be the right child of $n_m$;
15           Make the subtree rooted by $n_b$ as the right
16           subtree of $n_r$;
17           if $n_r$ originally has its right child $n_l$
18               Find a node $n_f$ in the subtree rooted by $n_r$
19               with $t_f = t_r$ and having either no middle or
20               no right child;
21               Insert the subtree rooted by $n_l$ as the
22               middle or right subtree of $n_f$;
```

Fig. 17. Summary of the storage adjustment process.

6. DEFECT TOLERANCE

In this section, we show how to extend the aforementioned temporal floorplanning algorithm to handle the defect tolerant issue. With the standard microfabrication techniques [Fair et al. 2003] and the synthesis result, a digital microfluidic biochip can be fabricated. However, due to the underlying mixing technology, the microfluidic biochips have unique defects and failure mechanism [Su et al. 2006]. The reconfigurability of the biochips can bypass the

```
Algorithm: Feasibility Detection ($H$)
$H$: a T-tree;
1 Set the durations of all storage units $v_s$ as zero;
2 repeat
3   if a clustered node $n_i$ is not the left child of node $n_j$
4       Move $n_i$ to the position of the left child of $n_j$;
5   else if a storage $n_s$ is not in its feasible position
6       Move $n_s$ to one of its feasible positions;
7   else if $n_s$ has its left child
8       Assume that $v_s$ stores the result of $v_a$;
9       Find a node $n_z$ in the subtree rooted by $n_a$ but not
10      in the subtree rooted by $n_s$'s left child $n_k$;
11      Move Subtree (the subtree rooted by $n_k$, $n_z$);
12  else
13      Traverse $H$ to obtain the starting time of
14      each task;
15      Check the precedence constraints and
16      reconstruct $H$;
17 until the topology of $H$ is fixed
18 Determine the duration of all storage units;
19 Storage Adjustment($H$);
20 Output: $H$ with the feasible topology.
```

Fig. 18. Summary of the feasibility detection and tree reconstruction process.

defective cells to tolerate the defects due to fabrication. Moreover, the non-reconfigurable device, such as the detectors, may be unavailable due to the existence of the defects. We need to rebind the operations from an unavailable detector to other available detectors. Note that after fabrication, the locations of the non-reconfigurable devices are fixed. We cannot move these detectors during floorplanning.

The central idea of our algorithm is that we model each defective cell as an obstacle. If a cell c located at (x, y) becomes faulty, we create an obstacle d_c located at (x, y) with its duration being the same as the assay completion time. In the packing process, we do not allow the overlap among tasks and obstacles. By this method, we can guarantee that no task will overlap with obstacle d_c, and thereby avoid to place tasks on defective cells.

We now present our obstacle avoidance algorithm to avoid overlaps among reconfigurable operations and obstacles. As mentioned above, we create an obstacle for each defective cell. During the packing process, we detect if a task v_i overlaps with any obstacle d_c. If v_i overlaps with an obstacle d_c, we first calculate the *X-span* s_x and *Y-span* s_y. The X-span (Y-span) represents how far we should shift v_i along the X (Y) direction to avoid the overlap with d_c. In this paper, we set s_x (s_y) as the difference between x'_c and x_i (y'_c and y_i), where (x'_c, y'_c) is the up-right coordinate of d_c. We shift v_i along the direction that results in smaller movement distance. That is, if $s_x < s_y$, we shift v_i in the X direction; otherwise, we shift v_i in the Y direction. Figure 19 summaries our obstacle avoidance process.

```
Algorithm: Obstacle Avoidance ($v$, $D$)
$v$: a task;
$D$: set of obstacles;
1 repeat
2   Scans all obstacles in $D$;
3   if $v$ overlaps with $d_c \in D$
4       Let $s_x$ be $|x_v - x'_c|$;
5       Let $s_y$ be $|y_v - y'_c|$;
6       if $s_x < s_y$
7           Set $x_v$ equal $x'_c$;
8       else
9           Set $y_v$ equal $y'_c$;
10          Update the $x$-coordinate of $v$;
11 until $v$ overlaps no obstacles in $D$;
```

Fig. 19. Summary of the obstacle avoidance algorithm.

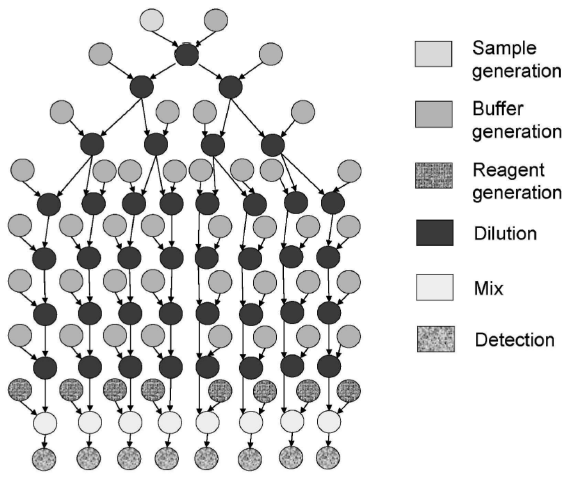


Fig. 20. The sequencing graph of colorimetric protein assay [Su and Chakrabarty 2005b].

7. EXPERIMENTAL RESULTS

Our algorithm was implemented in the C++ programming language and run on a 1.066 GHz SUN Blade 1500 machine with 4 GB memory. We implemented the unified synthesis and placement algorithm proposed in Su and Chakrabarty [2005b] and the 3D-subTCG representation on the same machine. For 3D-subTCG, we used the same SA engine as T-tree. We modified the operations of 3D-subTCG to satisfy the storage constraint at each perturbation. We also applied the clustering algorithm proposed in Section 5.1 to 3D-subTCG for fair comparison. For all experiments, we set $\alpha = \frac{1}{21.5}$, $\beta = \frac{0.5}{21.5}$, and $\gamma = \frac{20}{21.5}$. We also assumed that there exists one segregation cell between any two operations. All experimental results are the best result obtained by simulated annealing.

We evaluated our placement algorithm with two bioassays: the colorimetric protein assay [Srinivasan et al. 2004] and the multiplexed in-vitro diagnostics [Su and Chakrabarty 2004]. Figure 20 shows the sequencing graph of the

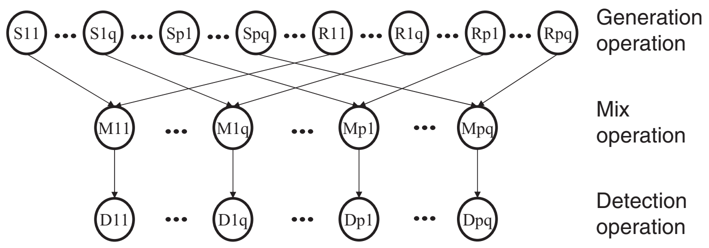


Fig. 21. The sequencing graph of multiplexed in-virto diagnostics [Su and Chakrabarty 2004].

colorimetric protein assay while Figure 21 shows the multiplexed in-virto diagnostics with p samples (Plasma, Serum, Urine, and Saliva) and q reagents (glucose, lactate, pyruvate, and glutamate). For the colorimetric protein assay, we applied the same design specification (resource constraint) and used the same microfluidic module library as Su and Chakrabarty [2005b]. We assumed that there is only one reservoir/dispensing port for sample fluid, two such ports for buffer fluid, two ports for reagent fluid, and one port for waste fluid. We also assumed that there are at most four optical detectors integrated on the biochip [Su and Chakrabarty 2005b]. For the multiplexed in-virto diagnostics, we used the same design specifications (resource constraint) as Su and Chakrabarty [2004]. We assumed that there is one reservoir/dispensing port for each type of samples and reagents and one optical detector for each enzymatic assay [Su and Chakrabarty 2004]. However, since [Su and Chakrabarty 2004] did not specify the width, height, and duration of each reconfigurable operation, we generated the areas/durations of each type of the mix operations based on the ratio of areas/durations of each reconfigurable operation in Su and Chakrabarty [2005b]. Table I shows the microfluidic module library used for the multiplexed in-vitro diagnostics.

First, we assumed that no defective cells exist. Table II summarizes the result of the colorimetric protein bioassay. Column 2 lists four different design specifications (fixed-cube constraints). We report the resulting volume (area times assay completion time) and CPU time (in seconds). As shown in this table, our algorithm can meet all design specifications (fixed-cube constraints) while both [Su and Chakrabarty 2005b] and 3D-subTCG cannot. More importantly, [Su and Chakrabarty 2005b] (3D-subTCG) requires, on average, 1.68X (1.61X) larger volume and 4.32X (39.96X) longer CPU time than our algorithm. Table III shows the result of the multiplexed in-vitro diagnostics. In this experiment, we used three examples for evaluation. Column 1 shows the number of types of samples and reagents, and column 2 lists the type of samples and reagents used in each example. For each example, we applied three different design specifications, as listed in column 3. We also report the volume and CPU time in this experiment. As shown in this table, our algorithm can meet all design specifications (fixed-cube constraints) while both 3D-subTCG and Su and Chakrabarty [2005b] cannot. Su and Chakrabarty [2005b] obtains larger volumes in all three examples (3.48X, 4.90X, and 3.84X) with longer CPU times (9.71X, 9.81X, and 19.19X, respectively); 3D-subTCG also obtains larger

Table I. The Microfluidic Module Library for the In-vitro Diagnostics

Operation	Resource	Duration (sec.)
Dispense	on-chip	2
Mix (plasma)	2 × 4-array	3
	2 × 3-array	6
	2 × 2-array	10
	1 × 4-linear array	5
Mix (Serum)	2 × 4-array	2
	2 × 3-array	4
	2 × 2-array	6
	1 × 4-linear array	3
Mix (Urine)	2 × 4-array	3
	2 × 3-array	5
	2 × 2-array	8
	1 × 4-linear array	4
Mix (Saliva)	2 × 4-array	4
	2 × 3-array	8
	2 × 2-array	12
	1 × 4-linear array	6
Opt (glucose)	LED + Photodiode	10
Opt (lactate)	LED + Photodiode	8
Opt (pyruvate)	LED + Photodiode	12
Opt (glutamate)	LED + Photodiode	10
Storage	single cell	N/A

Table II. The Experimental Result of the Colorimetric Protein Bioassay

		[Su and Chakrabarty 2005b]		T-tree	
Bioassay	Design Spec.	Volume	CPU Time (sec.)	Volume	CPU Time (sec.)
Protein	10 × 10 × 400	10 × 10 × 349	275.05	9 × 9 × 241	78.03
	10 × 10 × 360	9 × 10 × 339	270.02	10 × 9 × 211	57.27
	11 × 11 × 320	10 × 11 × 313	266.16	10 × 10 × 221	68.32
	9 × 9 × 400	(9 × 10 × 390)*	293.21	9 × 9 × 240	65.21
Average		1.68	4.32	1.00	1.00

		3D-subTCG		T-tree	
Bioassay	Design Spec.	Volume	CPU Time (sec.)	Volume	CPU Time (sec.)
Protein	10 × 10 × 400	10 × 10 × 239	2497.94	9 × 9 × 241	78.03
	10 × 10 × 360	10 × 10 × 331	2226.71	10 × 9 × 211	57.27
	11 × 11 × 320	11 × 11 × 272	4036.13	10 × 10 × 221	68.32
	9 × 9 × 400	(11 × 9 × 398)*	1984.04	9 × 9 × 240	65.21
Average		1.61	39.96	1.00	1.00

Volume = Area × Completion Time. ()*: the result cannot meet the design specification.

volumes in all three examples (2.50X, 2.05X, and 1.86X, respectively) with longer CPU times (38.52X, 23.89X, and 41.39X, respectively). The two experimental results clearly show the efficiency and effectiveness of our algorithm with different bioassays and design specifications. The results of 3D-subTCG also support our claim in Section 3.5 that the T-tree is a more suitable 3D representation for the placement problem of biochips. Figure 23 shows the placement

Table III. The Experimental Result of the Multiplexed *in-vitro* Diagnostics

			[Su and Chakrabarty 2005b]		T-tree	
Bioassay	Description	Design Spec.	Volume	CPU Time (sec.)	Volume	CPU Time (sec.)
in_vitro	S1, S2, S3, and S4	9 × 9 × 100	9 × 9 × 98	85.28	6 × 9 × 67	9.12
($p = 4$,	are assayed with A1,	8 × 8 × 120	(10 × 9 × 117)*	107.48	6 × 4 × 98	13.22
$q = 4$)	A2, A3, and A4	7 × 7 × 140	(9 × 9 × 126)*	118.65	7 × 4 × 96	10.17
Average			3.48	9.71	1.00	1.00
in_vitro	S1, S2, and S3 are	8 × 8 × 100	8 × 8 × 98	74.43	5 × 4 × 74	7.00
($p = 3$,	assayed with A1,	7 × 7 × 120	(7 × 9 × 112)*	84.51	6 × 4 × 62	8.28
$q = 4$)	A2, A3, and A4	6 × 6 × 140	(7 × 8 × 150)*	87.29	5 × 4 × 73	10.14
Average			4.90	9.81	1.00	1.00
in_vitro	S1, S2, and S3 are	7 × 7 × 80	7 × 7 × 79	46.16	4 × 4 × 60	3.63
($p = 3$,	assayed with A1,	6 × 6 × 100	(6 × 8 × 93)*	52.66	4 × 4 × 61	4.78
$q = 3$)	A2, and A3	5 × 5 × 120	(5 × 8 × 120)*	58.22	4 × 4 × 64	1.72
Average			3.84	19.19	1.00	1.00

			3D-subTCG		T-tree	
Bioassay	Description	Design Spec.	Volume	CPU Time (sec.)	Volume	CPU Time (sec.)
in_vitro	S1, S2, S3, and S4	9 × 9 × 100	9 × 9 × 97	474.43	6 × 9 × 67	9.12
($p = 4$,	are assayed with A1,	8 × 8 × 120	8 × 8 × 97	305.34	6 × 4 × 98	13.22
$q = 4$)	A2, A3, and A4	7 × 7 × 140	(6 × 9 × 135)*	411.37	7 × 4 × 196	10.17
Average			2.50	38.52	1.00	1.00
in_vitro	S1, S2, and S3 are	8 × 8 × 100	6 × 7 × 72	191.59	5 × 4 × 74	7.00
($p = 3$,	assayed with A1,	7 × 7 × 120	6 × 7 × 86	206.16	6 × 4 × 62	8.28
$q = 4$)	A2, A3, and A4	6 × 6 × 140	6 × 6 × 69	196.75	5 × 4 × 73	10.14
Average			2.05	23.89	1.00	1.00
in_vitro	S1, S2, and S3 are	7 × 7 × 80	5 × 6 × 60	102.20	4 × 4 × 60	3.63
($p = 3$,	assayed with A1,	6 × 6 × 100	6 × 5 × 58	166.79	4 × 4 × 61	4.78
$q = 3$)	A2, and A3	5 × 5 × 120	5 × 5 × 80	105.17	4 × 4 × 64	1.72
Average			1.86	41.39	1.00	1.00

()*: the result cannot meet the design specification. (S1: Plasma, S2: Serum, S3: Urine, S4: Saliva, A1: Glucose, A2: Lactate, A3: Pyruvate, A4: Glutamate).

result of the colorimetric protein assay with the 10 × 10 × 400 design specification. For simplicity, we only show the reconfigurable and detection operations.

Now we demonstrate the effectiveness of our algorithm for handling the defective cells. Assume that the biochip of Figure 23 is fabricated. Similarly to Su and Chakrabarty [2005b], we assumed that one optical detector is rendered defective due to fabrication. Therefore, the detection operations that were originally mapped to the defective detector must be re-mapped to other detectors. In this experiment, we set the fixed architecture as 9 × 9 and the limit of assay completion time as infinity. In this way, our algorithm can minimize the assay completion time while satisfying the design specification. Table IV lists the result of defect tolerance. Column 2 lists the locations of the defective cells. We considered four different cases with different number and location of defective cells. We report the assay completion time (in seconds) and the CPU time (in seconds). As shown in this table, our algorithm can obtain 16% longer

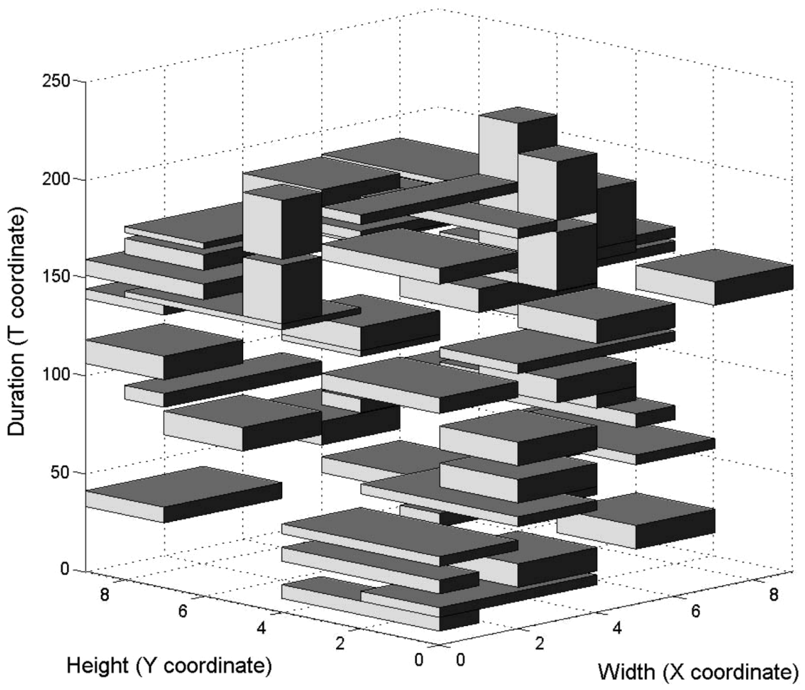
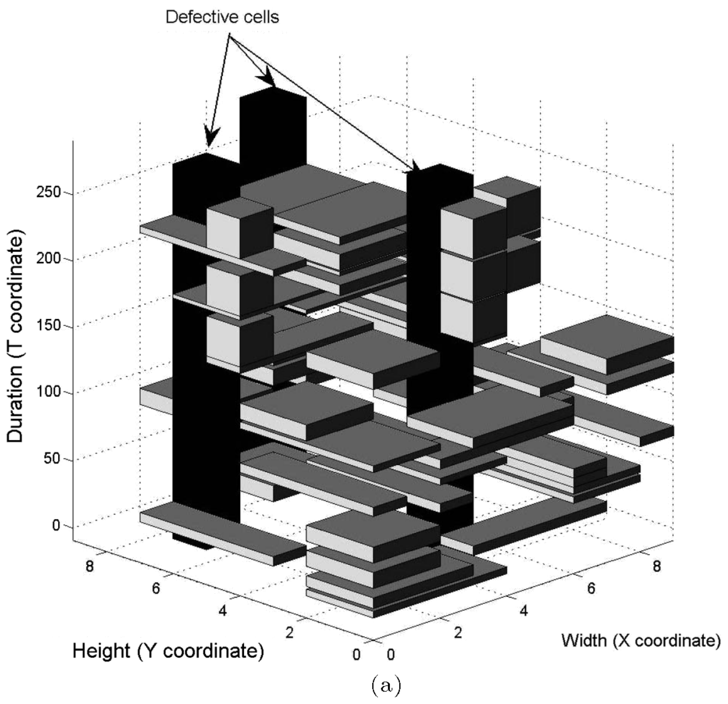


Fig. 22. The 3D view of the placement result of the protein bioassay with the $10 \times 10 \times 400$ design specification.

average assay completion time (280 vs. 241) with 42% longer average CPU time (111.46 vs. 78.03) than defect-free placement. This experimental result demonstrates that our defect-tolerance algorithm can operate a bioassay on a defective biochip with reasonable CPU time. Figure 23 shows the two placement results with three and four defective cells.

For the last experiment, we demonstrate the effect of our clustering method proposed in Section 5.1 on the protein bioassay. Table V shows the result of the protein bioassay with and without clustering. Columns 3 and 4 show the volume and CPU time without clustering and columns 5 and 6 show the volume and CPU time with clustering. As shown in this table, we can observe that the T-tree with clustering achieves 27% smaller volume and 6% less CPU time compared with the T-tree without clustering. The reduction on volume comes from the elimination of unnecessary storage units between generation operations and reconfigurable operations. Therefore, the SA engine can obtain a more compact 3D floorplan. The saving in CPU time is not as significant as the reduction on volume, because we do not actually cluster two operations into one operation. Moreover, we need extra CPU time to ensure that a clustered node is the left child of another clustered node during the tree reconstruction process. This result shows the effectiveness of the proposed clustering algorithm. The result also shows that it is important to make use of the properties of a bioassay during floorplanning.



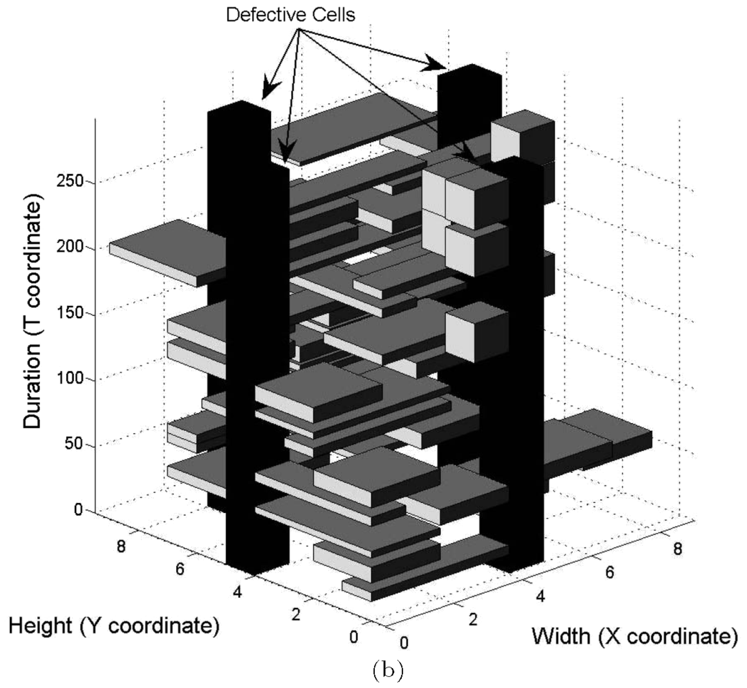


Fig. 23. (a) The 3D view of the placement result with three defective cells located at (4, 2), (1. 6), (5, 8). (b) The 3D view of the placement result with four defective cells located at (0, 4), (4, 0), (2, 7), (7, 5).

Table IV. The Defect Tolerance Result

Bioassay	Defective cells	Assay Completion Time (seconds)	CPU Time (seconds)
protein	(0, 4), (2, 6), (7, 7)	261	85.37
	(4, 2), (1, 6), (5, 8)	281	82.45
	(0, 4), (4, 0), (2, 7), (7, 5)	299	151,74
	(3, 0), (5, 2), (6, 3), (8, 4)	279	140.96
Average		280	111.46

Table V. The Experimental Result of the Colorimetric Protein Bioassay with and Without Clustering. Volume = area × completion time

		w/o Clustering		w/ Clustering	
Bioassay	Design Spec.	Volume	CPU Time (sec.)	Volume	CPU Time (sec.)
Protein	$10 \times 10 \times 400$	$10 \times 10 \times 249$	59.73	$9 \times 9 \times 241$	78.03
	$10 \times 10 \times 360$	$10 \times 10 \times 230$	104.97	$10 \times 9 \times 211$	57.27
	$11 \times 11 \times 320$	$11 \times 11 \times 262$	48.27	$10 \times 10 \times 221$	68.32
	$9 \times 9 \times 400$	$9 \times 9 \times 280$	72.45	$9 \times 9 \times 240$	65.21
Average		1.27	1.06	1.0	1.0

8. CONCLUDING REMARKS

In this article, we have applied the temporal floorplanning technique to the placement problem of digital microfluidic biochips. The motivation is that the physical placement of operations can be handled by 2D floorplanning techniques. Moreover, previous works show that floorplanning techniques are applicable to some scheduling problems, such as Xia et al. [2003] and Wuu et al. [2004]. Therefore, to simultaneously perform scheduling and physical placement, we model the placement problem of biochips as the temporal (3D) floorplanning problem. The advantage of this approach is that we have a high flexibility to optimize both the assay completion time and the biochip area (and other constraints, such as the defect tolerance requirement, as well).

To our best knowledge, our work is the first to adopt a topological representation (the T-tree representation) for the placement problem of digital microfluidic biochips. We have also proposed a clustering algorithm to cluster a generation operation and a reconfigurable operation to obtain a smaller volume and to reduce the CPU time. Due to the need to perform a bioassay on a biochip with the existence of defects, the proposed placement algorithm handles the defect tolerant issue by modeling each defective cell as an obstacle and not allowing overlaps among operations and obstacles. We have shown the efficiency and the effectiveness of our algorithm over previous works.

Future work lies in finding more sophisticated methods for handling the storage units as well as considering the fault tolerance issue and the design-for-defect/fault tolerance requirement during floorplanning. Another potential research direction lies in mapping the placement problem of biochips to other problems, instead of the floorplanning one. For example, the placement problem can be mapped to the unified high-level synthesis and physical design problem ([Dougherty and Thomas 2000; Gu et al. 2005]). A bioassay is represented as a control data flow graph. The goal is to optimize the schedule length (i.e., assay completion time) and area with the consideration of registers allocation (i.e., the placement of storage units). The placement problem may also be mapped to the 3D placement problem [Goplen and Sapatnekar 2003; Obenaus and Szymanski 2003]. A task is represented by a 3D cell. The placement problem of biochips is equivalent to the placement of 3D cells in the 3D space with a given 3D cube. The challenges of this approach are how to handle the precedence constraints and how to perform resource binding during placement, since resource binding changes the temporal ordering requirement among non-reconfigurable operations and the dimension of reconfigurable operations. Quantitative analysis would be needed to determine the best approach (floorplanning, placement, or unified high-level synthesis and physical design) for the placement problem of biochips.

REFERENCES

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

Cheng, L., Deng, L., and Wang, M. D. F. 2005. Floorplanning for 3-d VLSI design. In *Proceedings of Asia South Pacific Design Automation Conference*. 405–411.

Ding, J., Chakrabarty, K., and Fair, R. B. 2001. Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays. *IEEE Trans. Comput. Aid. Design Integrat. Circ. Syst. 20*, 1463–1468.

Dougherty, W. E. and Thomas, D. E. 2000. Unifying behavioral synthesis and physical design. In *Proceedings of Design Automation Conference*. 756–761.

Fair, R. B., Srinivasan, V., Ren, H., Paik, P., Pamula, V., and Pollack, M. 2003. Electrowetting-based on-chip sample processing for integrated microfluidics. In *In Proceedings of IEEE International Electron Device Meeting*. 32.5.1–32.5.4.

Fekete, S. P., Kóhler, E., and Teich, J. 2001. Optimal fpga module placement with temporal precedence constraints. In *Proceedings of Design, Automation and Test in Europe*. 658–665.

Goplen, B. and Sapatnekar, S. 2003. Efficient thermal placement of standard cells in 3d ics using a force directed approach. In *Proceedings of International Conference on Computer Aided Design*. 86–89.

Gu, Z. P., Wang, J., Dick, R. P., and Zhou, H. 2005. Incremental exploration of the combined physical and behavioral design space. In *Proceedings of Design Automation Conference*. 208–213.

ITRS. The international technoloy roadmap for semiconductors: http://public.itrs.net/.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science 220*, 4598 (May), 671–680.

Lin, J.-M. and Chang, Y.-W. 2001. TCG: A transitive closure graph-based representation for non-slicing floorplans. In *Proceedings of Design Automation Conference*. 764–769.

Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y. 1995. Rectangle-packing-based module placement. In *Proceedings of International Conference on Computer-Aided Design*. 472–479.

Obenaus, S. T. and Szymanski, T. H. 2003. Gravity: Fast placement for 3-d vlsi. *ACM Trans. Design Autom. Electr. Syst. 8*, 3, 298–315.

Ricketts, A. J., Irick, K., Vijaykrishnan, N., and Irwin, M. J. 2006. Priority scheduling in digital microfluidics-based biochips. In *Proceedings of Design, Automation and Test in Europe*. 329–334.

Srinivasan, V., Pamula, V., Paik, P., and Fair, R. 2004. Protein stamping for maldi mass spectrometry using an electrowetting-based microfluidic platform. In *Proceedings of the International Society for Optical Engineering*. 26–32.

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

Su, F. and Chakrabarty, K. 2005a. Design of fault-tolerant and dynamically-reconfigurable microfluidic biochips. In *Proceedings of Design, Automation and Test in Europe*. 1202–1207.

SU, F. AND CHAKRABARTY, K. 2005b. Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips. In *Proceedings of Design Automation Conference*. 825–830.

SU, F., CHAKRABARTY, K., AND FAIR, R. B. 2006. Micrpfluidic-based biochips: technology issues, implementation platforms, and design-automation challenges. *IEEE Trans. Comput.-Aid. Design Integrat. Circ. Syst. 25*, 4, 211–223.

TUTORGIG. http://www.tutorgig.com/encyclopedia.

WONG, D. F. AND LIU, C. L. 1986. A new algorithm for floorplan design. In *Proceedings of Design Automation Conference*. 101–107.

WUU, J.-Y., CHEN, T.-C., AND CHANG, Y.-W. 2004. Soc test scheduling using the b-tree based floorplanning technique. In *Proceedings of Asia South Pacific Design Automation Conference*. 1188–1191.

XIA, Y., CHRZANOWSKA-JESKE, M., WANG, B., AND JESKE, M. 2003. Using a distributed rectangle bin-packing approach for core-based soc test scheduling with power constraints. In *Proceedings of International Conference on Computer Aided Design*. 100–105.

YAMAZAKI, H., SAKANUSHI, K., NAKATAKE, S., AND KAJITANI, Y. 2000. 3d-packing by meta data structure and packing heuristics. *IEICE Trans. Fundam. Electr. Commun. Comput. Science E83-A*, 4, 639–645.

YUH, P.-H., YANG, C.-L., AND CHANG, Y.-W. 2004. Temporal floorplanning using the T-tree formulation. In *Proceedings of International Conference on Computer-Aided Design*. 300–305.

YUH, P.-H., YANG, C.-L., CHANG, Y.-W., AND CHANG, H.-L. 2004. Temporal floorplanning using 3d-subTCG. In *Proceedings of Asia South Pacific Design Automation Conference*. 725–730.

Received November 2006; revised February 2007; accepted April 2007