Fluid-to-cell assignment and fluid loading on programmable microfluidic devices for bioprotocol execution

流体加载是一种特殊类型的流体路径，它取决于单元的位置，其中流体将加载到FPVA芯片上。现有的FPVA生物芯片流体路由问题技术考虑了碰撞避免和可靠性问题，但没有考虑模块内必须装载流体的单元。假设单元的容量为单位体积，因此时序图中提到的所需流体体积必须占据相同数量的FPVA单元。

给定生物方案，为了在只有一个输入端口的FPVA芯片中加载不同的流体，需要创建一系列流体流动。如果这种流动的总数增加（从减少加载布线条数的方面优化），那么将需要更多数量的昂贵流体，并且还会增加芯片退化的风险。因此，将所有流体加载到 FPVA 芯片的不同单元中所需的流体数量是一个需要最小化的重要因素。最小化此流体流量以及确定流动路径称为流体负载问题。

由于流体加载问题包含了资源放置问题和避障路径问题，其中路径必须穿过一些预定义的单元，并且障碍物可以是任何直线多边形，因此流体加载问题显然是*NP*困难的。如图1（a）所示，如果两个单元格都有“*S*”（*S*类试剂），那么如果*Fj*（指的流体路径）出现在*Fi*之前，那么这种加载情况将得到解决。同样如图1（b）所示，如果这两个单元中的一个有“*S*”，另一个有“*B*”，则其中一个流必须绕道来解决这种情况。但是，如果我们有一个*Smap*（实际上是流体存放在组件位置的结果，前面两种情况都是单元的加载，图1（c）则是在一块区域或是组件中的加载），其情况与图1（c）所示相同，则两个流永远无法解决该情况。如果一个流动从一个单元流出，那么流动只能向它来自的方向以外的三个方向之一移动（这方面的确实应该考虑）。因此，在图1（c）中，如果最初填充了“*S*”，则放置在突出显示的单元格上的“*B*”无法填充任何有效的流量。同样，如果最初填充了“*B*”，则放置在突出显示的单元格上的“*S*”无法填充任何有效的流。因此，在这种情况下，即使流体的数量是两个，也必须有三个流体数量来加载它们。

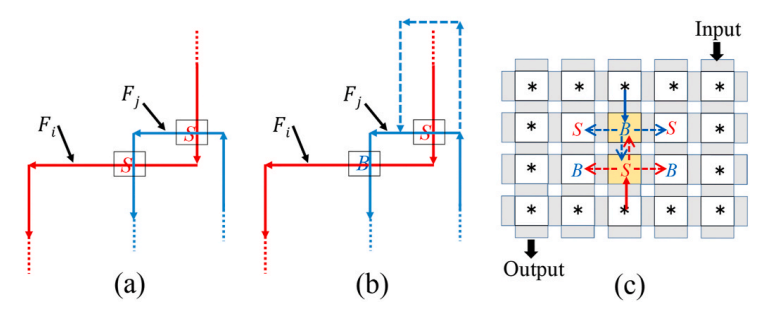


图1 流体加载问题示意图

考虑图2（a）和（b） 中所示的*Smap*，如果使用FLAP算法（图2（a）），则需要六个流，在最好的情况下，加载“*B*”和“*S*”至少需要三个流（图2（b））。同样，如果考虑图2（c）和（d）中所示的*Smap*，如果使用FLAP，则需要三个流，而在最佳情况的数量*K*能够减少到两个。

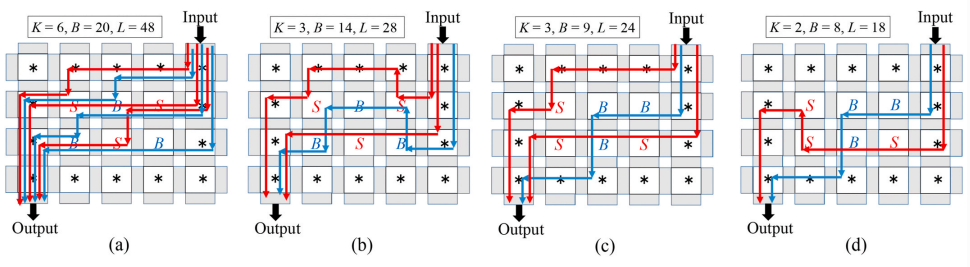


图2 （a）使用FLAP[1]的任意*Smap*所需的流数. （b）使用最优*K*的*Smap*. （c）使用FLAP[1]的另一个任意*Smap*所需的流数和（d）使用最优*K*的*Smap*。

文章提出了一个两阶段的方法来解决这个问题。

首先，提出了一种基于约束满足问题（CSP）的方法，称为负载感知流体单元分配（LAFCA），以获得合适的流体到单元分配，这将有利于第二阶段。

在第二阶段，即流体加载阶段，制定了另一种基于CSP的方法，称为基于CSP的加载算法（CSPLA），该算法生成一系列流动以完全解决流体加载问题。

FPVA芯片的流体到单元分配（LAFCA）的问题模型如下：

输入：*n* × *m* FPVA 和片上模块的所需位置 *M* = {*M*1， ...， *Mu*}，用于包含 *T* 种不同流体（试剂）的生物协议的调度图，由 {*t*1， *t*2， ...， *tT*} 表示，这些流体将以 *Mps* 为单位加载，*∀p*， *p* = {1， ...， *u*}。

输出：加载感知分配映射*Smap*。

目标：获得一个优化的*Smap*。

CSP 求解器可能无法在合理的时间内求解大型 CSP 实例。因此，除了 CSPLA 之外，还提出了一种有效的流体加载阶段启发式方法，称为确定最后流量 （DFL） 方法。DFL基于执行流的想法，该流尽可能最后加载最大期望的单元格，以便其他流获得最小的障碍。FPVA芯片的流体加载问题模型如下：

输入：作为*n* × *m* FPVA芯片的*Smap*的流体到单元分配。

输出：流体流动序列 *F* = {*F*1， *F*2， ...， *FK*}，它根据给定 *Smap* 的需要加载 FPVA 单元中的所有流体。

目标：找到具有最小流量数（在本例中为 *K* 个）的解决方案，该流量能够成功加载流体，如*Smap*中所示。

为给定的流体负载问题产生最优或接近最优的解决方案，同时考虑将流量 *K* 数作为最小化的单一目标。

CSPLA和DFL的目标是确定流动序列，以便它们将流体加载到所需的单元中，如图 3（a） 所示。流体流动将芯片上任何预先放置的流体视为障碍物，并以一种方式操纵其路径，使其永远不会与图3（b）和（c）中描述的任何此

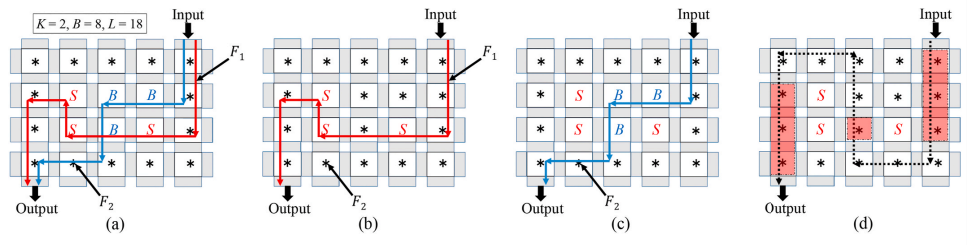


图3 流体各次序路径及清洗示意图

类障碍物发生碰撞。然而，在将任何流体加载到其所需的池中后，需要缓冲液来从其流路中的其他池中洗涤该流体，这个问题被称为流体洗涤问题。如图3（d）中加载流体*F*2后需要清洗的单元用红色显示。

CSP和CSPLA都采用求解器算法，这边跳过（无非是添加约束公式然后求解），主要介绍启发式的DFL算法。

DFL中使用的启发式背后的思想：将图4（a）中的流量*F*1*last*视为最后一个流量，即FLAP中的第一个流量。流动*F*1*last*在其路径上加载流体“*B*”。这意味着它在其路径上覆盖了单元中的流体（用绿线表示）。即，允许*F*1*last*之前的任何流体将任何类型的流体加载到*F*1*last*路径上的单元中。因此，在示例中，将“*L*形”区域（由绿色虚线包围）视为“\*”（障碍），如图4（b）所示，用于*F*1*last*之前的流动。通过这个考虑，我们有更多的自由度在*F*1*last*之前找到“好”的流程。

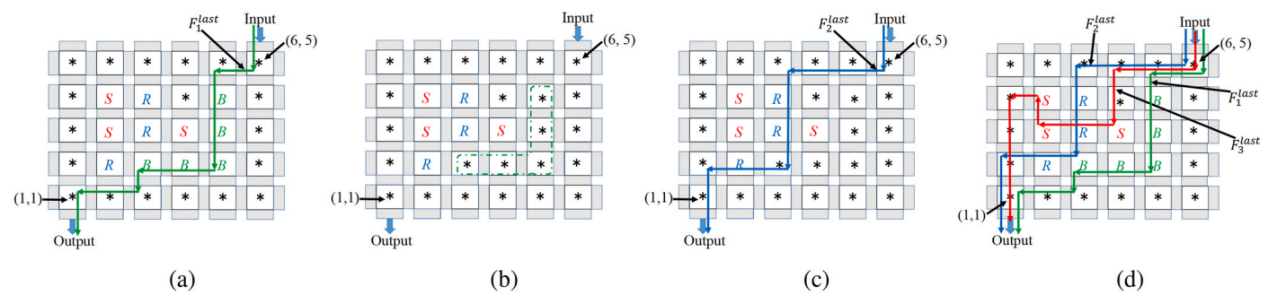


图4 DFL算法*Smap*示例的加载流程

基于这一考虑，总体策略如下：

•与常规方法不同，算法以相反的顺序确定流量。

•将已确定的流量上的单元格标签更新为“\*”，并找到一个“良好”流量，该流量在已确定的流量之前加载流体。其中“良好”的流动：包含尽可能多的流体单元，并且较短的路径会更好。

此后，使用 *Weightti*（*Fj*） 作为流动*Fj*路径上流体类型 *ti* 的单元数。通过选择 *Weightti*（*Fj*） 最高的流体数量来确定“最佳”流量（*BestFlow*方法）。

使用图5详细解释了*F*2*last*（“最佳”流量）的生成，如图4（c）所示，*F*2*last*用于流体“*R*”。*BestFlow*（R）方法找到图4（c）中用黑色箭头表示的流*F*2*last*，这是图14（a）中所示的*Smap*的倒数第二个流。在此示例中，*Cellstart* 是单元（2， 2），如图5（a） 所示，由流体“*R*”周围的圆圈表示。下一个任务是从*Cellstart*到输入端口单元（6， 5）找到一个包含尽可能多的“*R*”的流路径。从单元（1， 1）到*Cellstart*的路径在图5（a）中用红色箭头表示，因此不能将此路径用于输入端口的路径。因此，在这些单元格中放置了“*P*”标签，以将它们标记为从 *Cellstart* 到输入端口单元（6， 5）的任何流动路径的障碍物。

对于剩下的任务，算法的想法如下。每个单元都有四个传入边和四个出边，它们连接到其相邻的单元。如图5（a） 所示，每条边都有一个值，该值表示从 *Cellstart*开始的路径中包含的“*R*”的数量。起初，除了*Cellstart*的传出边之外，每条边的值都为0。来自*Cellstart*的传出边沿的值为1，该值将传播到输入端口。在传播值时，如果进入一个标签为*ti*（本例中为“*R*”）的单元格，则该值将增加1，因为路径现在会找到一个带有“*R*”的新单元格。具有“*R*”或“*\**”以外的标签的单元格应停止这种传播。在图5（a）中，无法遍历的单元格用红色的“*X*”表示。如果尝试所有可能的传播过程，将找到包含流体“*R*”的最大单元数的路径。

但如果尝试所有可能的传播过程，将找到包含流体“*R*”的最大单元数的路径。但是，这很耗时，因此算法仅在值增加时进行传播（*UpdateCells*（*ti*））。

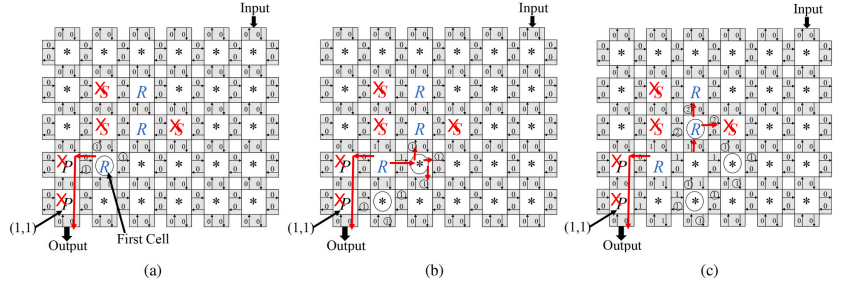


图5 在DFL方法中使用函数*UpdateCells*处理如图4问题实例. （a）初始情况. （b） *UpdateCells*的第一次调用. （c）*UpdateCells*的第二次调用.

如图5（b） 所示，在第一次调用*UpdateCells*时，只检查单元（2，1）和单元（3，2），它们与单元（2，2）（*Cellstart*） 相邻，并由“\*”周围的圆圈表示。这是因为在初始情况下，只有它们的入射边具有值1。*UpdateCells*在第一次调用时将单元（2，1） 和单元（3，2）的传出边的值更新为1。更新的值在图5（b）中圈出。在下次调用*UpdateCells*时，将更新三个单元格（3，3）、（3，1）和（4， 1）的传出边的值，这些值由图5（c）中的圆圈表示。传出边缘上的更新值由圆圈包围。单元（3，3）的传出边的值更新为2，因为从（1，1）开始并贯穿此单元格的流包含两个“*R*”。

在上述过程中，应该注意不要形成流循环。为了做到这一点，算法将路径信息与相应的边缘附加在一起。例如如图5（c）所示，考虑第二次调用*UpdateCells*后单元（3，1） 到（3，2）的传出边缘，此边的值刚刚更新为1，用图中的圆圈表示。此更新的值1对应于一个路径：{（2，2）、（2，1）、（3，1）}，该路径有一个“*R*”。我们将此路径信息存储在来自单元（3，1）的单元（3，2）的传入边缘。在图5（c）之后的第三次*UpdateCells*调用时，我们考虑更新（3，2） 的传出边，因为它具有来自单元（3，1）的更新传入边，而不会将传出边的值更新为单元（4，2）和（3，3），因为它们已经是1。此外，也不会将传出边上的值从当前值0更新为单元（2，2），因为当前传入边的路径信息，即{（2，2）、（2，1）、（3，1）}具有（2，2），这将创建一个流动循环，算法整体流程图如图6所示。

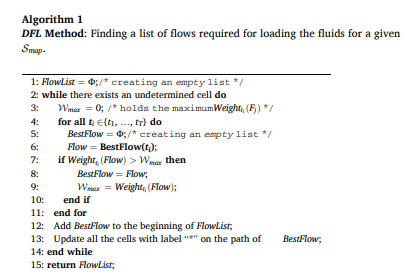


图6 DFL算法流程图s

[1] Gupta A, Huang J D, Yamashita S, et al. Design Automation for Dilution of a Fluid Using Programmable Microfluidic Device--Based Biochips[J]. ACM Transactions on Design Automation of Electronic Systems (TODAES), 2019, 24(2): 1-24.