Algorithms for Biological Cell Sorting with a Lab-on-a-chip

Arijit Ghosh, Rushin Shah
Computer Science and Engineering Department
Indian Institute of Technology, Kharagpur
Kharagpur, India.
Email: {arijitiitkgpster, rushin.shah}@gmail.com

Arijit Bishnu and Bhargab B. Bhattacharya
Advanced Computing and Microelectronics Unit,
Indian Statistical Institute,
203, B. T. Road, Kolkata, India - 700108.
Email: {arijit, bhargab}@isical.ac.in

***Abstract*—Automatic cell sorting and isolation for recovery of such live cells, mostly microorganisms, is a challenging task. Lab-on-a-Chip devices implemented as cell arrays are used for this purpose. For an abstract model of the problem, we can assume the cell array to be represented by a matrix where each cell can be any of the three types: empty, good (or desired) and bad (or undesired). The problem is to separate the good and the bad cells by pushing them under electric field to their respective receptors at the corners of the cell array. We address some combinatorial optimization problems related to this biological phenomenon.**

Keywords: algorithm, optimization, cell sorting, lab-on-a-chip.

I. Introduction

Rare cell population, e.g. adult stem cell, is available in very small quantities in samples that also have limited supply. Automatic cell sorting and isolation for recovery of such live cells is a challenging task. The method involves an enrichment step by magnetic or fluorescent cell sorting followed by manual or automatic cell picking or analysis [3]. Thus, applications in the medical, biological and pharmaceutical fields like stem cell research, cell therapy and cell based diagonistics need both microorganism detection and manipulation [4], [5], [6], [7], [8]. A Lab-on-a-Chip (LOC) is a device that can integrate several laboratory functions on a single chip of very small size. LOCs can handle very small fluid volumes. LOCs with sensing, processing and actuation functions can serve this purpose [1]. Microorganisms can be manipulated or displaced from their location using dielectrophoresis (or DEP). DEP is a physical phenomenon in which a force is exerted on neutral particles when it is subject to non-uniform electric field. The microorganisms can also be detected using DEP cage approach [2] and impedance sensing [1]. Differences in permittivity and conductivity between the particles and the suspending medium is used for detecting and then manipulating the microorganisms. The manipulation is done by applying electric fields using DEP. Static DEP cages have been developed that can trap live individual cells into closed potential cages [3].

To sum up the process of LOC, we have the power of detecting and manipulating or moving microorganisms. The microorganisms are manipulated using voltage differentials (electric fields) and moved so that they can be collected at desired locations. The samples are prepared on a cell array by culturing it with some reagent so that normal or desired samples can be differentiated from undesired ones. These are then manipulated using electric fields so that they are separated and trapped and collected at the corresponding DEP cages or receptors. By manipulation, we mean the microorganisms on the cell array are displaced towards their corresponding receptor. This way the cell array is exhausted of the microorganisms. Once the microorganisms are collected at their receptors, several biological tests may be performed on them. Thus, for an abstract model of the problem, we can assume the cell array to be represented by a matrix where each cell can be any of the three types: empty, desired and undesired. We have to empty the cell array by pushing the desired and undesired cells to their respective receptors.

A. Problem Definition

We are given a matrix $\mathcal{A}$ of $M \times N$ cells . Each such cell (i, j) $(1 \leq i \leq M,\ 1 \leq j \leq N)$ can have 3 possible values: Red (denotes as R), Blue (denoted as B) and Empty (denoted as E). Let N_r denote the number of R cells, N_b denote the number of B cells and N_e denote the number of E cells. So, $N_r + N_b + N_e = M \times N$. We denote the cell in the i^{th} row and j^{th} column as (i, j). For a cell (i, j), we define its neighbourhood to be the set of cells $\mathcal{N} = \{(i, j-1), (i, j+1), (i+1, j), (i-1, j)\}$ if (i, j) does not lie on the horizontal or vertical boundaries of $\mathcal{A}$. If (i, j) lies on the horizontal or vertical boundaries or the corners, then the neighborhood of (i, j) is an appropriate subset of $\mathcal{N}$, e.g., the neighborhood of $(1, N)$ is $\{(1, N-1), (2, N)\}$ and the neighborhood of $(2, N)$ is $\{(1, N), (2, N-1), (3, N)\}$.

A cell with value E can exchange its value with any of its neighbouring cells. A cell with value R or B can exchange its value only with a neighbouring E cell. Such exchanges are performed with the help of a voltage differential applied across such neighbouring cells. This can be done by using a manhattan wire layout beneath the cells. Such exchanges allow a cell to reach to its corresponding receptor.

An R-receptor is located outside the matrix, adjacent to the cell $(1, 1)$. Any R in this cell is removed by the R receptor, and hence can be replaced in cell $(1, 1)$ by an E. Similarly, a B receptor is located outside the matrix, adjacent to the cell $(1, N)$. Any B in this cell gets removed by the B-receptor, and

is replaced in cell $(1, N)$ by an E. So, the number of E cells increases in the matrix as the emptying process goes on.The R and B receptors provide a mechanism for emptying R and B cells from the matrix. See Figure 1 for an example. Henceforth,

$$R \sqsupset \begin{pmatrix} E & B & B & R \\ B & R & R & B \\ B & B & R & B \\ R & R & B & R \end{pmatrix} \sqsubset B$$

Fig. 1. An example of a cell array with R-receptor and B-receptor. Here, $M = 4, N = 4, N_e = 1, N_r = 7, N_b = 8$.

we shall omit representations of the R and B receptors and implicitly assume their existence unless stated otherwise. Our goal, in this paper, is to minimize the number of exchanges, and hence, the number of voltage differentials, to empty the cell array.

In Section II, we discuss the existence and lower bounds of this problem. Section III discusses a heuristic to solve the problem of emptying all cells from the cell array using the minimum possible number of transitions.

II. Emptying cells with minimum transitions

Refer Figure 2. For any cell (i, j) depending on whether it is red or blue, the minimum distance that the cell has to travel to reach its corresponding receptor is at least the length of the monotonic manhattan path from the cell (i, j) to the receptor as shown in Figure 2. Let $\ell(i, j)$ be the length of such a path from the cell (i, j). The length is measured in terms of the number of cells visited. Now, while emptying the matrix, the manhattan path that the cell takes may not be monotonic. This happens because cells of other color act as obstacles. Let the length of this path be $\ell'(i, j)$. Surely, $\ell'(i, j) \geq \ell(i, j)$. The length of such manhattan paths is directly proportional to the number of voltage differentials applied for such movements. Our optimization problem is to minimize the number of such voltage differentials. Thus, we

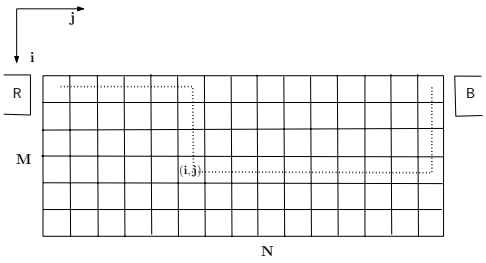

Fig. 2. A schematic of a cell array with the coordinate axis shown.

look into the following problem.

Problem 1: Let $\ell'(i, j)$ be the length of the manhattan path taken by a cell (i, j) to reach its corresponding receptor. Our problem is to minimize $\sum_{(i,j)\in\mathcal{A}}(\ell'(i, j))$.

We first probe the existence of such paths and try to find a lower bound on N_e, the number of empty cells required for a strategy to empty the matrix. We define a *grid graph* $G = (V, E)$ on the cell array as follows. Each cell $(i, j) \in \mathcal{A}$ represents a vertex and there is an edge between (i, j) and the neighbors $\mathcal{N}$ as defined earlier. Let $G_{V\setminus v}$ denote a graph obtained from G by deleting the vertex $v \in V$ and the edges incident on v.

Definition 1: Given a graph $G(V, E)$ if $G_{V\setminus v}$ remains connected $\forall\ v \in V$ then it has the reduced connected property.

We have the following simple observation from graph theory.

Observation 1: Given a $M \times N$ grid graph G, where $M, N \geq 2$. The graph $G_{V\setminus v}$ is reduced connected.

This follows from a simple fact that in a $M \times N$ grid graph G, where $M, N \geq 2$, neighbours of any vertex $v \in V$ remains connected in $G_{V\setminus v}$. We now have a lemma about the existence of a strategy.

Lemma 1: For any cell array matrix $\mathcal{A}$ of size $M \times N$ with $M, N \geq 2$, there always exists a strategy for emptying $\mathcal{A}$ if $N_e \geq 1$.

Proof: We show that given a red cell R_{ij} at the position (i, j), we can move it to $(1, 1)$ if there is at least one empty cell in the matrix. We use induction on $(i + j)$.

Basis Step: Let $i + j = 2$. That implies $i = 1$ and $j = 1$ as $i, j \geq 1$. Hence, it is already at $(1, 1)$.

Induction hypothesis: The assumption is true for $i + j = n$, where $n \geq 2$.

Induction Step: Let $i + j = n + 1$. Since, $n \geq 2$, at least one of i, j is greater than equal to 2. Let us assume without loss of generality, $j \geq 2$. Let G be the grid graph on $\mathcal{A}$. Now, consider $G_{V\setminus v}$ where v is the vertex corresponding to the cell at (i, j). Note that, the cell at (i, j) is not an E cell. $G_{V\setminus v}$ remains connected because of Observation 1. So, there exists a path from an E cell to the cell at $(i, j-1)$. We can perform exchanges along this path to bring the E cell to the location $(i, j-1)$ such that the path does not go through R_{ij} (the R cell at (i, j)). After the E cell is brought to the location $(i, j-1)$, we can exchange it with R_{ij}. Now the R cell at (i, j) has moved to the location $(i, j-1)$. Now, $i+j-1 = n$. From our induction hypothesis, we know that for a red cell at location (h, k), where $h + k \leq n$, we can shift it to $(1, 1)$. Similarly, we can show a result for any B cell to move to the location $(1, N)$.

Once we have moved the R cell to the location $(1, 1)$, we can empty it and still $N_e \geq 1$. This shows that at least one E cell is required for an emptying strategy to exist. Hence, the result is proved by induction. ■

Henceforth, we assume for simplicity $M = N$, i.e. $\mathcal{A}$ is an $N \times N$ matrix.

Lemma 2: Lower bound on the minimum number of exchange operations required to empty any square matrix of dimensionality $N \geq 2$ is $\Omega(N^3)$.

Proof: Let $\mathcal{A}$ be a square matrix of dimensionality N. We denote by e_{ij} the value of the cell in the matrix $\mathcal{A}$, which can take values R or B or E. Now, given the initial arrangement of the matrix $\mathcal{A}$, if $e_{ij} = \mathsf{R}$, then the cell at (i, j) has to make atleast $i + j - 2$ transitions to reach $(1, 1)$ so that it can be emptied by a Red receptor and $N - 1 + i - j$ transitions to reach $(1, N)$ cell so that it can be emptied by the Blue receptor

if $e_{ij} = \mathsf{B}$. Let OPT_N be lower bound on the minimum number of operation required to empty any square matrix of dimensionality N.

$$\begin{aligned} OPT_N &\geq \sum_{e_{ij}=R} (i+j-2) + \sum_{e_{ij}=B} (N-1+i-j) \quad (1) \\ &= \sum_{e_{ij}=R \ or \ B} (i-1) + \sum_{e_{ij}=R} (j-1) + \\ &\quad \sum_{e_{ij}=B} (N-j) \quad (2) \end{aligned}$$

Note the subscripts of the summation sign in Equation 1; it is an either-or sum. Without any loss of generality we can assume that number of nonempty cells is of the order $\Omega(N^2)$. This is a valid assumption as we are trying to derive a lower bound of the minimum number of operations required to empty any given matrix and the value of OPT_N will only increase with the increase in number of non-empty cells. With this assumption, we have,

$$\sum_{e_{ij}=R \ \ or \ \ B} (i-1) = \Omega(N^3) \quad (3)$$

Putting the value from eqn(3) to eqn(2), we get

$$\begin{aligned} OPT_N &\geq \sum_{e_{ij}=R \ \ or \ \ B} (i-1) + \sum_{e_{ij}=R} (j-1) + \\ &\quad \sum_{e_{ij}=B} (N-j) \\ &= \Omega(N^3) + \sum_{e_{ij}=R} (j-1) + \sum_{e_{ij}=B} (N-j) \\ &= \Omega(N^3) \\ &\quad \big(\text{as } \sum_{e_{ij}=R} (j-1) \geq 0 \text{ and } \sum_{e_{ij}=B} (N-j) \geq 0\big) \\ OPT_N &= \Omega(N^3) \quad (4) \end{aligned}$$

Therefore, $OPT_N = \Omega(N^3)$. ■

III. A Heuristic for Emptying the Matrix

Thus, we have shown that the complexity of the problem of emptying all the cells from the matrix has a lower bound $\Omega(N^3)$. There are many algorithms that succeed in performing the task with asymptotically optimal time complexity. However, we want to find an algorithm that achieves the task by using minimum possible transitions. In that effort, we present below a heuristic to that end.

A. Non-Optimal Transitions

For a 2×2 subset of some given matrix, where X represents the "don't care" condition and can be R or B or E, consider the following transformations:

$$T_1 \equiv \begin{pmatrix} B & R \\ x & E \end{pmatrix} \rightsquigarrow \begin{pmatrix} R & B \\ x & E \end{pmatrix}$$

$$T_2 \equiv \begin{pmatrix} B & R \\ E & x \end{pmatrix} \rightsquigarrow \begin{pmatrix} R & B \\ x & E \end{pmatrix}$$

These transformations prove that any RB duo can be transformed to a BR duo and vice-versa using only 1 adjacent E cell, and without disturbing any other cell.

We will only show the details for transformation T_1 and T_2 follows similarly.

Sequence of transitions in T_1 when $x = R$

$$\begin{pmatrix} B & R \\ R & E \end{pmatrix} \longrightarrow \begin{pmatrix} B & E \\ R & R \end{pmatrix} \longrightarrow \begin{pmatrix} E & B \\ R & R \end{pmatrix} \longrightarrow \begin{pmatrix} R & B \\ E & R \end{pmatrix} \longrightarrow \begin{pmatrix} R & B \\ R & E \end{pmatrix}$$

, and for the case $x = B$

$$\begin{pmatrix} B & R \\ B & E \end{pmatrix} \longrightarrow \begin{pmatrix} B & R \\ E & B \end{pmatrix} \longrightarrow \begin{pmatrix} E & R \\ B & B \end{pmatrix} \longrightarrow \begin{pmatrix} R & E \\ B & B \end{pmatrix} \longrightarrow \begin{pmatrix} R & B \\ B & E \end{pmatrix}$$

So we can see that transitions made in T_1 is different in both the cases but it is mentioned as one transformation so as to reduce the number of notations used in the proofs.

Definition 2: We define a non-optimal transition to mean moving an R cell such that its distance from the R receptor increases, or a B cell such that its distance from the B receptor increases. Any transition that is not a non-optimal transition is defined as an optimal or greedy transition.

If no non-optimal transitions occur in an algorithm that solves the problem, then by definition, such an algorithm would be optimal. However, we now prove that in some problem configurations, non-optimal transitions are unavoidable. We state the following result which follows trivially:

Result 1: T_1, T_2 transformations contain non-optimal transitions.

This follows from the fact that the sum total of the distances moved by cells towards their respective receptors were 2 but total transitions made were 4. Hence there were non-optimal transitions in T_1 and T_2.

Lemma 3: Some matrices cannot be emptied without non-optimal transitions.

We shall refer to collections of cells bounded by a B cell at the left and an R cell at the right in the 1^{st} row simply as a BR duo, since to empty these cells we need to make at least one non-optimal transformation. Thus, even the best possible algorithm for solving this problem, i.e. the optimal algorithm would still have to make a few non-optimal transitions for some matrices. Our goal is to keep the number of non-optimal transitions to minimum as well keep the running time of the algorithm in $O(N^3)$, where N is the dimensionality of the square matrix.

We now consider the question of whether some matrices are easier to empty than others. We note another result which will be used in the algorithm:

Result 2: Once the 1^{st} row has been emptied, the rest of the rows can be emptied optimally.

We provide a simple intuitive argument for this result. First, any cell in the 2^{nd} row can be brought to the 1^{st} row by

directly swapping it with the E cell above it in the 1^{st} row, and can then be brought to its receptor by successively swapping it with E cells. Once the 2^{nd} rows is emptied in this manner, any cells in the 3^{rd} row can be brought up to the 1^{st} row by swapping it with the 2 E cells above it, and then emptied as described above. In this manner, all the cells in all the rows can be emptied. Since all the transitions here are greedy transitions, and move any coloured cell strictly closer to its receptor, we can see that all the rows from the 2^{nd} row onwards can be emptied using greedy transitions.

Accordingly, we restrict ourselves to considering only the 1^{st} row when considering the difficulty of emptying any given matrix. Here, we must note that once the cells of any one particular colour have been removed, the 1^{st} row only consists of E cells and cells of the other colour, and hence the 1^{st} row can now be entirely emptied using greedy transitions as well.

B. The Heuristic

We use the following data structures in our algorithm: We are given an $N \times N$ matrix. Numbering starts from 1 for both rows and columns, i.e. (i, j) refers to the cell in the i^{th} row and j^{th} column. *ER*, *EB* are two variables that keep track of the last E cell of the consecutive E cells from the R or B receptors respectively in the 1^{st} row. Initially, both of these pointers are set to NULL. *Cost* is an integer variable that indicates the number of transitions observed till the present time, i.e. the cost incurred during the execution of the algorithm so far.

We now define some sub-routines that are used in the control loop of our algorithm:

CheckBoundary: This sub-routine is an initialization step. It checks if the $(1, 1) =$ R, then it makes $ER = 1$ and $(1, 1) = E$. Similarly, if $(1, N) =$ B, it makes $EB = 1$ and $(1, N) = E$. Any transition from R or B to E is spontaneous, so the value of *Cost* remains unchanged.

ExpandGreedily: In this subroutine, we first remove all the B, R cells in the 1^{st} row that can be emptied without making any non-optimal transitions. Once that is done, we reset the pointers ER and EB. After this step is done, whatever non-empty cells that remains in the 1^{st} row cannot be emptied without making non-optimal transitions. We push the remaining R, B cells in the 1^{st} row to the left, right respectively of the 1^{st} where ever possible using optimal transitions.

RemoveNonOptimally: It is assumed that the 2^{nd} row contains atleast one E cell otherwise we bring one to the 2^{nd} in such a way that tries to keep the increase in *Manhattan distance* to minimum. Once that is done we resolve the conflicts in the first row using the non-optimal T_1, T_2 transformations. For each optimal and non-optimal transition we add $+1$ amd $+4$ to the $Cost$ respectively.

ClearNextRows: Once the 1^{st} row is empty, remaining rows can be emptied optimally. Refer Result 2.

IsEmpty(row_{max}, col_{max}): This sub-routine will check if the sub-matrix with rows from 1 to row_{max} and columns from 1 to col_{max} is empty or not. If it is then it returns $True$ else $False$.

EmptyOptimally: We first try to empty as many cells as possible in a greedy manner by first calling the sub-routine CheckBoundary followed by ExpandGreedily. If all the cells are successfully emptied this way, the algorithm terminates and returns *Cost*. If the algorithm does not terminate, that implies there are BR conflicts in the 1^{st} row that needs to be converted to RB. We alternately call sub-routines RemoveNonOptimally and ExpandGreedily, until the 1^{st} row has been emptied. We then call the function ClearNextRows and empty the remaining cells in the other rows optimally. The algorithm then terminates and returns *Cost*.

Having estabilished these sub-routines, we now present our main algorithm:

Algorithm 1 EmptyOptimally

```
CheckBoundary
ExpandGreedily
if IsEmpty(N, N) then
   return Cost
end if{If the whole matrix has been emptied, return the cost}
SelectForNonOptimality
repeat
   RemoveNonOptimally
   ExpandGreedily
until IsEmpty(1,N) {Check if the 1st row has been emptied}
ClearNextRows
return Cost
```

Algorithm 2 SelectForNonOptimality

```
SR ⟵ Sum of distances of R cells in 1st row from R receptor
SB ⟵ Sum of distances of B cells in 1st row from B receptor
if SR ≤ SB then
   return R
else
   return B
end if{Whichever colour has the lesser sum should be eliminated using non optimal transitions so that such transitions are minimized}
```

The colour to be used by the RemoveNonOptimally function explained above is decided by this SelectForNonOptimality function.

Result 3: It is obvious that if the first row is empty or contains no BR conflicts, then our algorithm empties the matrix optimally.

This result has a practical significance. If the first row is washed away before the process or is treated as a waste, then the rest of the emptying process can be done optimally.

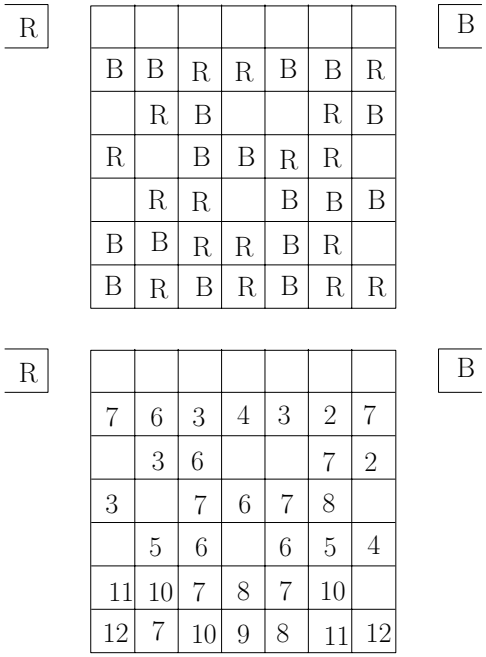

Fig. 3. Example of an optimally separable cell array with the number of moves $= \sum_{\forall(i,j)\in|B|\cup|R|} \ell'_{i,j}$. The numbers in the lower array shows the number of exchanges a corresponding cell in the upper array has to undergo to reach its corresponding receptor. The total number of exchanges is 229.

C. Analysis of the Algorithm

The idea that the algorithm uses is that once we have been able to clear the first row of the matrix then we can remove rest of the elements with the number of transitions equal to the sum of their *Manhattan distances* from the respective receptors.

To do the analysis of the algorithm we need to define few quantities

Definition 3: The quantities S_R, S_B is equal to the number of R, B cells remaining after clearing the first row in a greedy manner respectively.

We have already proved in Lemma 2 that minimum number of steps to empty any matrix is $\Omega(N^3)$ where N is the dimensionality of the square matrix.

For a given matrix M,

$$OPT_M \geq \sum_{\forall i,j} l_{ij} + 2min(S_R, S_B), \tag{5}$$

where l_{ij} is the *Manhattan distance* of each non-empty cell from its respective receptor and $2min(S_R, S_B) = O(N)$ in the worst case.

For our algorithm in the worst case, each R cell has moved through all the B cells, and the number of B cells remaining after greedily clearing the first row is S_B, and each T_1 and T_2 transitions take two extra moves. And if we have only E cell in the worst case, then we have to move E each time we have to do the T_1 or T_2 transitions, and in worst case end up changing the manhattan distance of all cells in the lower row by 1. And number of transitions made by E other than the one involved in T_1 and T_2 were $\leq N \times S_R$. So the total extra moves made over the sum of the Manhattan distances of the cells from the receptor were

$$\begin{aligned} ExtraMoves &\leq 2S_BS_R + O(N^2) + O(N) + NS_R \\ &= O(N^2). \end{aligned} \tag{6}$$

Let the number of exchanges performed by our algorithm be T_{algo} and

$$T_{algo} = \sum_{\forall i,j} l_{ij} + ExtraMoves \leq \sum_{\forall i,j} l_{ij} + O(N^2).$$

We have already shown $\sum_{\forall i,j} l_{ij} = \Omega(N^3)$ and the extra moves are bounded by $O(N^2)$.

Result 4: Our algorithm returns a solution that is within $O(1)$ of the optimal solution and thus it is a constant factor approximation algorithm.

IV. Conclusion

In this paper, we introduced the problem of moving red and blue cells in a matrix with empty spaces. We proved that it was possible to empty the entire matrix, no matter what its initial configuration, provided there was at least one empty cell in it initially. We showed a heuristic for this task running in $O(n^3)$ time. We defined non-optimal transitions, and proved that for certain matrix configurations, non-optimal transitions were unavoidable. This algorithm can be applied in many practical domains, most obviously in biological cell sorting. We would like to address the problem of solving with minimum number of transitions. We could not solve the optimality for this minimum number of transitions, neither do we have a hardness proof as yet. We achieved a constant factor approximation algorithm for the optimization problem of minimizing the number of exchanges stated in Problem 1. We are also studying the problem of finding the equivalence of two such matrices. Given two cell arrays with different configurations but same distribution of the number of different colored cells in the array, whether one can be transformed into the other by a sequence of moves. The previous work done in this area indicates that any two configurations with the same distribution of cells of two different colours can be transformed into each other by a sequence of moves so that all intermediate configurations are connected [9].

References

[1] G. Medoro, N. Manaresi, A. Leonardi, L. Altomare, M. Tartagni, and R. Guerrieri, "A Lab-on-a-Chip for Cell Detection and Manipulation", IEEE Sensors Journal, vol. 3, no. 3, pp. 317-325, 2003.

[2] G. Medoro, N. Manaresi, M. Tartagni and R. Guerrieri, "CMOS-only sensor and manipulation for microorganisms", in proc. IEDM, pp. 415-418, 2000.

[3] A. B. Fuchs, A. Romani, D. Freida, G. Medoro, M. Abonnenc, L. Altomare, I. Chartier, D. Guergour, C. Villiers, P. N. Marche, M. Tartagni, R. Guerrieri, F. Chatelain and N. Manaresi, "Electronic sorting and recovery of single live cells from microlitre sized samples", Lab on a Chip, RSC Publishing, vol. 6, pp. 121-126, 2000.

[4] X-B. Wang, Y. Huang, P. R. C. Gascoyne and F. F. Becker, "Di-electrophoretic manipulation of particles", IEEE Trans. on Industry Applications, vol. 33, no. 3, May/June, 1997.

[5] G. Medoro, N. Manaresi, M. Tartagni and R. Guerrieri, "CMOS-only sensors and manipulators for microorganisms", Electron Devices Meeting, 2000, IEDM Technical Digest, pp.415-418, 2000

[6] R. Lovell-Badge, "The future for stem cell research", Nature, 414, pp. 88-91, 2001.
[7] S. Archer, T. T. Li, A. T. Evans, S. T. Britland, H. Morgan, "Cell Reactions to Dielectrophoretic Manipulation", Biochemical and Biophysical Research Communications, vol. 257, Issue 3, pp. 687-698, April, 1999.
[8] T. Muller, G. Gradl, S. Howitz, S. Shirley, Th. Schnelle, G. Fuhr, "A 3-D microelectrode system for handling and caging single cells and particles", Biosensors and Bioelectronics, vol. 14, Issue 3, pp. 247-256, 15 March 1999.
[9] A. Dumitrescu and J. Pach, "Pushing squares around", Graphs and Combinatorics, vol. 22, no. 1, pp. 37-50, 2006.