Pushing points and Grid Optimization Characterization and Algorithms

Synopsis of a Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology
in
Computer Science and Engineering

by

Shah Rushin Navneet

03CS3012

under the guidance of

Dr. Arijit Bishnu

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur
May, 2008

Chapter 1

The Red-Blue Problem

1.1 Motivation

Many real-world problems can be modeled as matrices consisting of C different types of cells. We refer to such matrices as C coloured matrices. In addition to the C colours, a cell might also be of no colour at all, a situation where we refer to the cell as being empty. For all such matrices with various values of C, of particular interest to us is the case $C = 2$. For example, indicator dye injected into the human bloodstream might attach itself to certain specific types of cells, while leaving others unattached. In image processing, grayscale images consist of white and black pixels only. Cells may also correspond to memory locations in storage devices.

There are many tasks we wish to perform efficiently in these situations, and this problem corresponds to finding efficient algorithms for manipulating the cells of such matrices. Such manipulations might include rearranging the cells of a matrix to achieve homogeneity, or emptying cells of a particular colour without affecting those of other colours, using specially designated receptor cells. We consider the problem of emptying all cells from a 2-coloured matrix, using the minimum possible number of transitions.

1.2 Problem Definition

We are given an $M \times N$ matrix of cells. Each such cell can have 3 possible values:

- R - Red
- B - Blue

- E - Empty

Let N_r denote the number of red cells, N_b denote the number of blue cells and N_e denote the number of empty cells.

Then $M \times N = N_r + N_b + N_e$

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.

We denote the cell in the i^{th} row and j^{th} column as (i, j).
The cell $(1, 1)$ is called a R receptor. Any R in this cell can be instantaneously replaced by an E.

The cell $(1, N)$ is called a B receptor. Any B in this cell can similarly be replaced by an E.

These replacements can hence be thought of as emptying an R or B from the matrix.

Within this framework, we examine a number of interesting problems, as described below:

We consider the problem of finding a lower bound on the number of E cells required in the matrix, in order to ensure that all the non - E cells can be emptied. We prove that this lower bound is $N_e = 1$, i.e. even if the matrix contains just 1 empty cell, it is possible to empty all the R and B cells from it, using this empty cell.

The proof for this particular lower bound also implies a strategy to empty all the coloured cells from the matrix. However, this strategy doesn't account for the possibility that emptying cells might become progressively easier as the number of E cells in the matrix increases. Hence, in order to devise an optimal algorithm, we must consider a strategy that takes advantage of the inherent structure of the problem.

It is our intuition that the problem of emptying all coloured cells from the matrix can be optimally accomplished in $O(n^3)$ time. We produce a proof for this particular upper bound.

Having obtained such a proof, we state an algorithm that empties the entire matrix, which in addition to having optimal asymptotic time complexity, also minimizes the number of cell transitions required to achieve its task. We also prove the opti-

mality of this algorithm.

In addition, we have discussed our implementation of this algorithm, and its application to various matrices. We start with a fixed number of empty cells N_e and vary the matrices according to the difficulty of emptying them greedily. We then repeat this procedure over different values of N_e. We present and discuss our results.

1.3 N_e required to empty the matrix

We sketch our proof that $N_e = 1$ suffices to empty all the coloured cells in the matrix:

Suppose there exists only 1 E cell in the $M \times N$ matrix.

We know that E can shift to any cell in the matrix. Without loss of generality, assume it is in position $(1, 1)$.

Now, suppose we have a given configuration as follows:

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

The B receptor is blocked by an R cell. The R cell at position (M, N) is separated from the E cell at $(1, 1)$ by only B cells. If we can show a procedure to remove this R cell before removing any other cell, we can use this procedure to empty any R or B cell from the matrix, no matter how many opposite coloured cells separate it from the E cell.

Consider the following series of transitions:

$$\frac{RB}{xE} \rightarrow \frac{RE}{xB} \rightarrow \frac{ER}{xB} \rightarrow \frac{BR}{xE}$$

Thus any RB duo can be converted to a BR duo and vice-versa using only 1 adjacent E cell.

We denote any transformation of this type as $RB \leftrightarrow BR$

Using this result, we show that an R at (M, N) i.e. at maximum possible separation from $(1, 1)$, and separated from $(1, 1)$ entirely by B cells, can still be emptied using just 1 E cell.

Every cell in the matrix can be emptied using this procedure. In fact, in many cases, a coloured cell will have a clear path of E cells to its receptor, and will not be separated thus by opposite cells, and emptying it will require fewer transitions. Thus the procedure we have shown accounts for the worst case scenario.

1.4 Complexity of emptying the matrix

For a cell at position (i, j), at least $\sqrt{i^2 + j^2}$ transitions will be required to empty it. Thus, the minimum number of transitions required to empty all such cells is given by:

$$S = \sum_{i=1}^{M} \sum_{j=1}^{N} \sqrt{i^2 + j^2}$$

The order of this double summation S is the worst case lower bound of the problem of emptying all cells from the matrix.

Without loss of generality, let us assume that $M = \theta(N)$

Let $S' = S + T$, where $T = \sum_{i=1}^{N} \sqrt{i^2 + i^2}$

Now, $S \geq N^2$, so adding T to S will not change the order of S.

Hence $S = \theta(S')$.

Now, $S \leq \sqrt{2} \sum_i \sum_j \sqrt{max(i, j)^2}$

$S \leq 2\sqrt{2}n * n + (n - 1) * (n - 1) \ldots$

$S = 2\sqrt{2} \sum n^2$

$S = O(n^3)$

Hence, $S = O(n^3)$

Similarly,

$S' \geq \sqrt{2} \sum_{i,j} \sqrt{min(i,j)^2} + \sum_{t=1}^{N} \sqrt{t^2 + t^2}$

$S' = 2\sqrt{2}n * 1 + 2 * (n-1) + 3 * (n-2) \ldots$

$S' = o(\sum r(n-r))$

$S' = o(n^3)$

Thus, $S = o(n^3)$ and we proved ealier that $S = O(n^3)$

Hence, $S = \theta(n^3)$

Thus, we have shown that the problem of emptying all the cells from the matrix has a complexity of $\theta(n^3)$. There are many algorithms that will succeed in performing the task with this time complexity. However, we want to find the most optimal of all such possible algorithms, i.e. one that achieves the task by using minimum possible number of transitions. We present such an algorithm in our work.

1.5 Insight into Optimality

Let us define a non-optimal transition to mean moving an R cell away from the R receptor or a B from its receptor.
If we show that no non-optimal transitions occur in our algorithm then we have proved that the algorithm will be optimal.

However, we prove a lemma that in some problem configurations, non-optimal transitions are unavoidable.

Lemma 1: some matrices cannot be emptied without $RB \leftrightarrow BR$ transformations. e.g. consider the 2 row matrix shown below:

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

Thus in this case, in order to empty the cells, we need to make some $RB \leftrightarrow BR$ transformations.

Lemma 2: some non optimal transitions are unavoidable in some cases.
For the matrix presented in Lemma 1, in order to empty its cells, we need to make some $BR \leftrightarrow RB$ transformations. However, according to the definition of these transformations, they include non-optimal transitions. Hence, in some cases, non-optimal transitions are required.

Hence to prove the optimality of our algorithm, we must now prove that the number of non-optimal transitions is minimized during the execution of the algorithm. We have managed to do so.

1.6 Implementation of our Algorithm

We implemented our algorithm in C++. The user specifies the size of the matrix, as well as the number of R, B and E cells. The matrix can either be generated randomly, or entered by the user.

There is only one initial condition, namely that there exists at least one E in the matrix, and specifically that it is in the second row. This is necessary for a subroutine of our optimal algorithm. Since an E can move to any cell in the matrix in $O(n)$ time, we can assume this initial condition without loss of generality.

At each pass of the algorithm, the intermediate matrices are computed, including versions after non-optimal removal and after greedy removal. The total cost for removing all the cells is stored in memory.

Intuitively, matrices in which R cells are closer to the R receptor and B cells to their receptor must be easier to empty, than matrices in which it is the other way round. Accordingly, we informally define the following classes of matrices:

- Easy
- Medium
- Hard

The costs of emptying various sample 4×4 matrices in each of these cases are listed in Table 1:

Index	N_r	N_b	Cost
Easy 1	6	9	28
Easy 2	8	7	30
Medium 1	6	9	32
Medium 2	8	7	33
Hard	8	7	41

Table 1.1: Results for 4 x 4, $N_e = 1$

We repeated this procedure for similar matrices with $N_e = 1$, however of size 7×7. For these examples, we obtained the following results:

Index	N_r	N_b	Cost
Easy	27	21	188
Medium	26	22	208
Hard	21	27	230

Table 1.2: Results for 7 x 7, $N_e = 1$

Next, we ran our program on other 7 x 7 matrices, with varied values of N_e. The empty cells are interspersed throughout the matrix in question.

Similarly, we can obtain matrices for higher values of N_e. We consider matrices of dimension 7 x 7, for cases from $N_e = 5$ to $N_e = 15$. We ran our program on randomly generated matrices, for various values of N_r, N_b, N_e. We present our results in the thesis.

Our results indicate that the value of N_e, i.e. the number of empty cells in the matrix does not have a significant impact on the number of transitions needed to remove all the cells. The ratio of R cells to B cells also plays a role in determining the cost of emptying the matrix. If they are evenly balanced, the cost tends to be higher. There is a significant impact of the number of E cells only if multiple such cells happen to be in the first row. Moreover, we can also see, based on comparing the costs of emptying 4×4 versus 7×7 matrices that the number of transitions grows as $O(n^3)$, thus offering experimental validation of the algorithm's complexity.

Chapter 2

Linear Programming Problem

2.1 Problem Definition

We are given a $N \times N$ lattice. Each point (i, j) of the lattice may be either ON or OFF. If it is OFF, its associated value is said to be 0, and if it is ON, its associated value is said to be 1. Initially, a sequence of points from this lattice is set as ON. We wish to compute a new sequence of ON points which utilizes as few ON points as possible, and is subject to the following constraints:

- The boundary points are the same as those of the original sequence.
- The error compared to the given sequence is minimized.
- The point sequence is irreducible.
- The change in slope between successive points of the sequence is bounded.

We wish to formulate this problem as one of linear programming.

Now, since the lattice points can only take the values 0 or 1, this problem, once formulated as an instance of linear programming, will actually be an instance of integer linear programming, which is known to be NP-hard. Hence, we must initially ignore this constraint, and allow the point to take any real value between 0 and 1.

Once the problem has been formulated as an instance of linear programming, we obtain a solution to it using a standard mathematical software package, e.g. Mathematica. We then round off the values of the points of this solution to either 0 or 1, to obtain a solution to our original problem. There will be an error in the solution thus obtained, and hence we solve various example point sequences, and try to obtain an upper bound on this error by analyzing the errors produced in these individual point

sequences.

2.2 Formulation as linear programming

If a point (i, j) is initally ON, let $B_{ij} = 1$, else let $B_{ij} = 0$.
Let the two boundary points of the initial sequence be denoted by (i_l, j_l) and (i_h, j_h).
Hence $B_{i_l j_l}$ and $B_{i_h j_h}$ are necessarily equal to 1.
If a point (i, j) is ON in the computed sequence, let $X_{ij} = 1$, else let $X_{ij} = 0$.

We elaborate on the constraints mentioned earlier on the target point sequence:

Boundary:

The target sequence should have the same boundary points as the original sequence. Therefore, $X_{i_l j_l}$ and $X_{i_h j_h}$ must be equal to 1.

Error:

Let error E be defined as $-\sum_{i,j} B_{ij} X_{ij}$.

We can see that E as defined above measures the distance between the original and computed point sequences. $B_{ij} X_{ij} = 1$ only if the point (i, j) is ON in both sequences, 0 otherwise. Each such point that has differing states in the sequences contributes to the error E by an increment of 1.

Irreducibility:

Define every point of a sequence that is not a boundary point to be an interior point.
Define the neighbourhood of a point (i, j) to be the set of points $\{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$.

For a point sequence to be irreducible, each interior point of this sequence must have no more than 2 ON points in its neighbourhood, and each boundary point of the sequence must have no more than 1 ON point in its neighbourhood.

Intuitively, if a point sequence is not irreducible, there exists a smaller point sequence that can convey the same amount of information.

Bounded change in Slope:

Define the slope of 2 points $p = (i_1, j_1)$ and $q = (i_2, j_2)$ to be $S_{2,1} = (j_2 - j_1)/(i_2 - i_1)$. When we say that the change in slope at any point $p = (i, j)$ in the sequence is bounded, we mean:

$\mid (S_{p+1,p} - S_{p,p-1}) \mid \leq \beta$, where $p + 1$ is the next point and $p - 1$ is the previous point in the sequence, and β is some predefined bound.

However, this is not a linear equation. We must state these constraints in an equivalent linear form. We define the unacceptable slope transitions as those transitions that involve a change of slope $\beta > 2$. For example, $(0, 0) - > (1, 1) - > (0, 2)$ is an acceptable transition, while $(0, 0) - > (2, 1) - > (0, 2)$ is not. Hence, only 2 out of these 3 points may be ON, and the corresponding slope constraint for this unacceptable transition is:

$X_{00} + X_{21} + X_{02} \leq 2$

Therefore, we must enumerate such constraints for every combination of 3 points from the n^2 points of the given lattice that produces a slope change $\beta > 2$.

We are now ready to formally state the objective function and constraints in order to state this problem as an instance of linear programming:

Let the objective function be the minimization of $\sum_{i,j} X_{ij}$.

The constraints are:

$X_{i_l j_l} = 1$ and $X_{i_h j_h} = 1$

$E = -\sum_{i,j} B_{ij} X_{ij} < \tau$, where τ is some predefined threshold.

$\sum_{k=-1}^{1} X_{i_l - k j_l - k} \leq 2$

$\sum_{k=-1}^{1} X_{i_h - k j_h - k} \leq 2$

If (i, j) is not a boundary point, $\forall (i, j), \sum_{k=-1}^{1} X_{i-kj-k} \leq 3$

$X_{00} + X_{21} + X_{02} \leq 2$

$X_{00} + X_{31} + X_{12} \leq 2$

$X_{11} + X_{31} + X_{12} \leq 2$

$X_{00} + X_{33} + X_{03} \leq 2$

$\vdots$

$\forall (i_1, j_1), (i_2, j_2), (i_3, j_3)$ such that $\mid ((j_2 - j_1)/(i_2 - i_1)) - ((j_3 - j_2)/(i_3 - i_2)) \mid > 2$, $X_{i_1 j_1} + X_{i_2 j_2} + X_{i_3 j_3} \leq 2$

Analysis of constraints

Thus the program has been formulated as an instance of linear programming. There are:

- 2 boundary constraints.
- 1 error constraint.
- $O(n^2)$ reducibility constraints. There is one such constraint for each point, and there are n^2 number of points.
- $O(n^6)$ slope constraints. Each slope constraint involves a combination of 3 points, and there are n^2 such points, and 3 out of these n^2 points can be chosen in $n^2 C_3 = O(n^6)$ ways. We must choose all such combinations that result in a slope transition > 2, and it is clear that there are at least as many unacceptable transitions as acceptable ones, hence the number of unacceptable transitions, i.e. the number of slope constraints is also $O(n^6)$.

Hence, for any given lattice and initial sequence of points, we can enumerate the various reducibility constraints in $O(n^2)$ time, and the various slope constraints in $O(n^6)$ time. Thus we obtain a formal representation of the problem as an instance of linear programming.

2.3 Solution using Mathematica

In the previous section, we outline a procedure to convert a given lattice and point sequence into an equivalent linear programming problem. Once the problem is in this form, it can be solved by any computational mathematics package, such as Mathematica or Matlab, by using standard algorithms. For our task, we used Mathematica. We wrote a standard linear programming implementation with the variables, function and constraint as mentioned above, and ran this code on random sequences of various

sizes and initial point distributions.

For each such sequence, we obtain a set of solutions X_{ij}. However, here each element X_{ij} is a real number in the range $[0, 1]$. Hence, we round off each such X_{ij} to its nearest integer, since one of the constraints of the original problem, which we relaxed in order to convert the problem from integer linear programming to linear programming, is that $\forall(i, j), X_{ij} = 0$ or 1. We thus obtain the resultant point sequences.

2.4 Error Analysis

We compute the errors for each of the resultant point sequences as compared to their respective original point sequences. Recall that the error is defined as:

$E = -\sum_{i,j} B_{ij} X_{ij}$

We perform error analysis, to obtain a bound on this error, as a function of the input constants and the variables.