Branch and Bound Method for Discrete Tomography Reconstruction of hv-convex Binary Matrices

Miklós Póth

Polytechnical Engineering College, Subotica
Marka Oreškovića 16, 24000 Subotica, Serbia, email: pmiki@vts.su.ac.yu

Abstract: In this paper we show a new method for reconstructing binary matrices. We focus on the reconstruction of hv-convex matrices. We show that our algorithm finds all the solutions, and sort out the ones that do not comply to the hv-convex criteria. We also show that this algorithm works best for smaller matrices up to size 20x20 pixels. We make some notes on the possible speed-up strategys too.

Keywords: Binary Matrices, Discrete Tomography, Reconstruction, Branch and Bound

1 Introduction

1.1 Discrete Tomography

In discrete tomography we are concerned with reconstruction of matrices from few (in our case two) projections. We take the horizontal and vertical projections of the binary matrix that is shown in Figure 1.

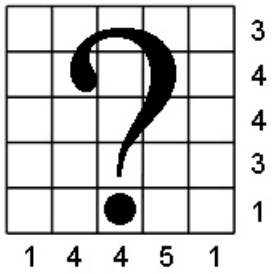


Figure 1
Example of a binary matrix to be reconstructed from two projections

In our case, we have

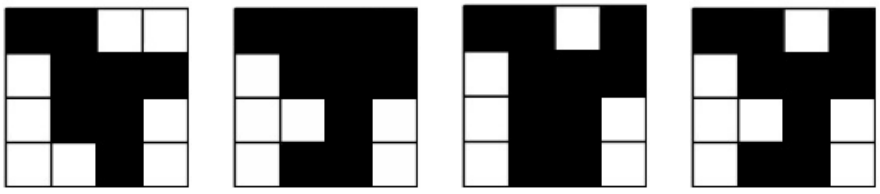
$$R = \begin{bmatrix} 3 & 4 & 4 & 3 & 1 \end{bmatrix} \tag{1}$$

$$C = \begin{bmatrix} 1 & 4 & 4 & 5 & 1 \end{bmatrix} \tag{2}$$

where ***R*** represents the row sum vector, and ***C*** represent the column sum vector. If a solution exists, we can reconstruct the binary matrix based on these two vectors. The conditions for existence and uniqueness were studied by Ryser [3]. The trivial condition for the existence of the result is that the sum of entries in ***R*** and ***C*** are equal.

1.2 hv-Convex Matrices

In this paper we are concerned with the reconstruction of hv-convex matrices. In such matrices the series of 1's is not broken in any of the rows or columns. This additional constraint considerabely reduces the number of possible solutions because no switching elements can occur in the entire matrix. Examples for matrix convexity are given in Figure 2.



a) b) c) d)

Figure 2
a) hv-convex matrix, b) h-convex matrix, c) v-convex matrix, d) no-convex matrix

1.3 Branch and Bound

Branch and bound algorithms are methods for global optimization. A large number of real-world planning problems called combinatorial optimization problems share the following properties: They are optimization problems, are easy to state, and have a finite but usually very large number of feasible solutions. The majority of these problems share the property that that no polynomial method for their solution is known. Branch and bound (B&B) is by far the most widely used tool for solving large scale NP-hard combinatorial optimization problems. B&B is, however, an algorithm paradigm, which has to be filled out for each specific problem type, and numerous choices for each of the components exist. However, branch and bound algorithms can be slow. In the worst case they require effort that grows exponentially with problem size, but in some cases we are lucky, and the methods converge with much less effort. Since the search space is intractably large, some

implicit method is required in order to rule out regions of the search space that contain no interesting solutions. Branch and bound algorithms work by the divide and conquer principle: the search space is subdivided into smaller subregions (this subdivision is referred to as branching), and bounds are found on all the solutions contained in each subregion under consideration. The strength of the branch and bound approach comes when bounds on a large subregion show that it contains only inferior solutions, and so the entire subregion can be discarded without further examination.

2 Explanation of the Method

The basic idea is to develop a tree in complyence with the content of the rows of the binary matrix. During tree development some nodes of the tree will not satisfy either the column sum condition, or the hv-convexity criteria. These "agak" will be cut out, narrowing the solution space. We find a solution every time we reach the leave of the tree.

Let's consider a 4-by-4 binary matrix with row sum

$$R = \begin{bmatrix} 2 & 3 & 2 & 1 \end{bmatrix} \tag{3}$$

and column sum

$$S = \begin{bmatrix} 1 & 3 & 3 & 1 \end{bmatrix} \tag{4}$$

Since we wolud like to reconstruct a hv-convex matrix, from $R(1) = 2$ we see that the 1's of the first row can be placed in three different positions, as shown in Figure 3. Generally, if we have to place k 1's on n positions, we have $(n-k+1)$ possibilities to do it. (If we want to reconstruct a matrix with no convexity constraint, we can place k 1's on n positions in n over k ways).

Figure 3
Three possibilities to place two 1's on four positions

Since no column sum violation occurred, we eliminate neither of the three nodes. We continue with the development of the tree in a similar way on the next (second) level. The structure of the tree on level 2 is shown in Figure 4.

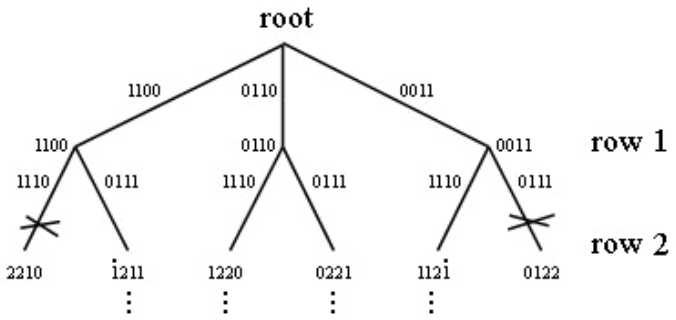


Figure 4
The tree after two levels of development

The four digit number at the nodes shows the current column sum. We now examine if any of the four digits exceed the given column sum. If so, we cut that edge, and we don't expand the tree further from that node. As we see from Figure 4, the leftmost and the rightmost nodes fail to satisfy this condition (2210 and 0122 exceeds the column sum 1331 on the first and last digit respectively), and those nodes won't be further expanded. This is one source of the efficiency of the method.

Since we aim to reconstruct a hv-convex matrix, we know that in any column of the matrix after a 1-to-0 transition no 1 can follow. This is illustrated in Figure 5. To track the occurance of 1-to-0 transitions, we put a marker (dot in Figure 4) on the digit where the transition occurred. This marker means that on that position no further 1 is possible. This measure also contributes to the efficiency of the method because it cuts out edges (nodes) narrowing the solution space.

Figure 5
If a black pixel occurs behind a 1-to-0 transition, the hv-convexity condition fails

The number of levels of the tree equals the number of rows of the matrix to be reconstructed (assuming that a solution exists). The number of solutions equals the number of leaves.

3 Performance Analysis and Speed-up Strategys

3.1 Matrix Rotation

Let's consider a binary matrix with the following row and column sum

$$R = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$C = \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}$$

After only two levels of development the tree will have 12 nodes from which only one (the last one) can be eliminated (Figure 6). The tree behaviour shows exponential growth, and that has great impact on computational time.

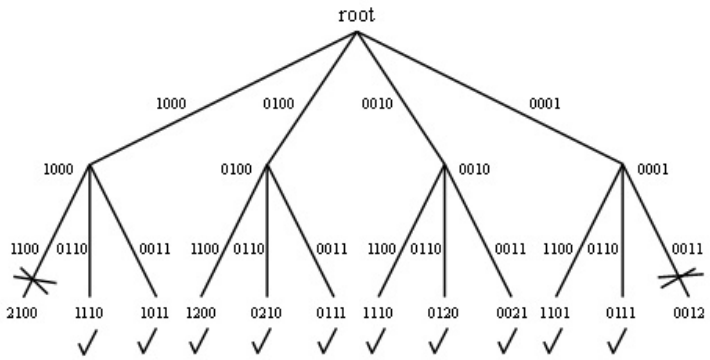


Figure 6
The tree after two levels of deevlopment. Only two edges can be cut

With matrix size increase, this growth becomes even more drastic. If the first two entries of the row sum matrix is 1 and 2, on the second level of a 100x100 matrix we'll have 9900 nodes (not necesserily all valid), and in such a case it would be very difficult and time consuming to process the matrix. One possible solution for the above problem could be matrix rotation. After a 90° CW rotation of the above matrix, *R* and *C* becomes:

$$R = \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}$$

After this small manipulation the tree becomes much simpler and has only 10 edges (Figure 7).

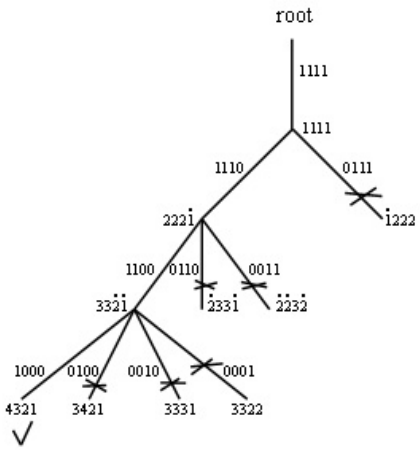


Figure 7
The tree is significantly simplified after the matrix rotation. Dot over digits shows that a 1-to-0 transition occured earlier on that path.

In this case reconstruction must be followed by 90° CCW rotation of the resulting matrix.

Such savings are not always possible, but rotation very often results in considerable speed-up.

3.2 Row Sum Maximum Start

However, rotation does not necesserily help to increase the computational speed. Very often, the majority of the 1's is placed in the middle of the matrix, and we have very low numbers next to the borders. The following matrix is a good example.

$$R = [2 \quad 2 \quad 6 \quad 8 \quad 7 \quad 3 \quad 2 \quad 1]$$

$$C = [2 \quad 3 \quad 3 \quad 6 \quad 7 \quad 6 \quad 3 \quad 1]$$

In this case, no rotation can prevent the tree from having at least 42 nodes at level 2, and even more nodes on lower levels. A possible solution to this problem could be to start from the row with most 1's, and to step one row up or down, depending on the number of 1's in those two rows. Doing so, we are certain that the tree will be minimal respecting the number of ones in each row, but we have to memorize an extra vector that will tell us the order of the rows. This is illustrated in Figure 8.

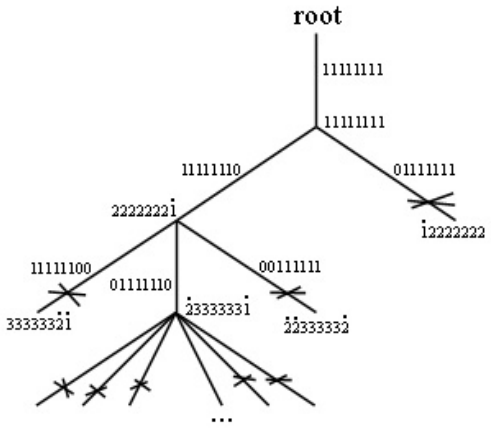


Figure 8
Developing the tree from with most 1's

3.3 Worst Case Scenarios

The worst case appears when in all rows we have only one entry. In that case for an *n*-by-*n* matrix on the first level there will be *n* edges, on the second level there will be *n-1* edges, and so on. It is easy to see that we'll have *n!* leaves on the last level. This is an extremely hard and time consuming problem.

4 Experimental Results

During the experimental phase, it was verified that our method finds all the possible solutions that satisfy the column sum constrints and the hv-convexity constraint. With minor changes it is possible to skip the hv-convexity constraint, and to reconstruct all the matrices that satisfy only the column sums. For the following matrix

$$R = [2 \quad 3 \quad 2 \quad 1]$$

$$C = [1 \quad 3 \quad 3 \quad 1]$$

our program gives the following solutions (Figure 9):

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 9
All the solutions for the above matrix

If we don't have the hv-convex condition, all the solutions are still valid, but we might have some extra solutions. If the above case, apart from those four solutions, we have two more (Figure 10).

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 10
Two more solutions if only the column sum condition has to be fulfilled

Conclusions

In this paper we showed a new method for reconstructing binary matrices from its orthogonal projections. We introduced the branch and bound method for discrete tomography problems. By putting a constraint of hv-convexity to the matrix to be reconstructed, the majority of the developed tree can be cut before we get to the bottom leaves. Experimental results show that this method finds all the possible results that satisfy the row sum and column sum constraint. We investigated two possible speed-up possibilities. The first speed-up strategy used matrix rotation, while in the second strategy we started from the row that was mostly populated with 1's. On some test matrices rotation performed better, on other matrices the second speed-up strategy showed better results. So before reconstruction, it is suggested to analyze the matrix first, and according to the elements of R and C to decide which method to use.

Acknowledgement

The preparation of this paper would not have been possible without the support, suggestions and ideas of professor Kálmán Palagyi.

References

[1] Marek Chrobak, Christoph Dürr: Reconstructing hv-Convex Polyominoes from Orthogonal Projections, Information Processing Letters, March 1999, Volume 69, Issue 6

[2] Richard Brualdi: Algorithms for Constructing (0,1)-Matrices with Prescribwd Row and Column Sum Vectors, Discrete Mathematics, Volume 306, Issue 23, pp. 2989

[3] H. J. Ryser: Matrices of Zeros and Ones, Bull. Amer. Math. Soc. Volume 66, Number 6 (1960), pp. 442-464

[4] H. J. Ryser: Combinatorial Properties of Matrices of Zeros and Ones, Canad. J. Math. Vol. 9 (1957) pp. 371-377

[5] Maciej Gebala: The Reconstruction of Convex Polyominoes from Horizontal and Verical Projections, Lecture Notes In Computer Science; Vol. 1521, 1998, pp. 350-359

[6] Alberto Del Lungo: Reconstructing Permutation Matrices from Diagonal Sums, Theor. Comput. Sci. 281(1-2), 2002, pp. 235-249