**Part I**

The pentomino problems can be considered as an Exact Cover Problem where all the different placement of each pentomino tile at different orientation is a subset of the entire board. In other words, we can choose some of the pentomino tiles from the subset and cover the entire board.

A mathematic representation can be a matrix where each row represents a subset. Each column of the matrix represents a particular cell in the matrix. Values can be assigned to the matrix to identify the existence of tiles in the cell of the board. In this problem, “0” was chosen to represent an empty cell, while positive integer was used to represent the occupied cell.

In addition, more columns need to be added to specify the tile identity since they can only be used once. A “1” was added at the ith column of the matrix. An illustration of the matrix is shown in the following figure.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Represent the cell occupied by the tile | | | | | Represent the identity of the tile | | |
| 1 | 1 | … | 0 | 0 | 1 | … | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| … | | | | | … | | |
| 0 | 0 | … | 12 | 12 | 0 | … | 1 |

To populate the matrix, all orientation of a tile is explored. For each orientation, all possible position of the tile is tested. The cells that are occupied by the possible placement are marked with a positive integer, and the entire board is flattened. The identity of the tiles is appended at the end of the placement 1-d array. This procedure is performed for all possible placement. For the 6x10 board, there are in total of 72 columns and 2057 rows.

To find a collection of the subset to satisfy the problem, we need to find rows in the matrix such that in the new matrix formed by stacking the rows, each column has only one positive integer. Algorithm X, proposed by Knuth, is effective in solving the problem. This is a recursive, back-tracking algorithm. Each time the function is called,

1. It first selects a column with the smallest number of positive integers. This column is deleted.
2. For every positive integer element in the column, it will delete the row of such element and add the row index to the solution.
3. For each positive integer in the delete row, the column such element is in will be deleted.
4. Lastly, rows contain a positive integer element in the deleted column will also be deleted.

The recursive call terminates successfully when there is nothing to be deleted.

To back-track, the algorithm will simply “undo” the deletion and choose another row with a positive integer to delete (step 2).

Each time the algorithm is run, constraints are applied in order to yield the solution. The deletion operation in the algorithm is essentially applying the constraints to reduce the number of possible solutions. Once a row is chosen (step 1), all other rows intersecting (have positive integer element) with the chosen row should not be in the solution. Therefore all such rows are deleted, and the size of the matrix is now smaller.

The heuristic used in the algorithm is the number of positive integers. The column with the smallest number of positive integers is chosen first since essentially there will be fewer possible outcomes. In other words, the back-tracking tree structure is smaller, making the running time faster.

A dancing link (DLX) data structure was used to represent the sparse matrix. This is a double-linked, four-directional, circular mesh of nodes. The manipulation of the pointers at each node made the operation very easy and fast.

References:

1. <https://arxiv.org/pdf/1010.5890.pdf>
2. <https://arxiv.org/abs/cs/0011047>