Knuth's Dancing Links

Oisín Hodgins, Seán Monahan

NUIG Final Year Project (Mathematics)

1 Introduction

1.1 The Exact Cover Problem

Given a set X, and a collection of its subsets S, an **exact cover** is the sub-collection S^* of S in which each element of X appears in exactly one subset of S^*.
In an **exact cover problem** we are concerned with determining whether an exact cover exists. The most famous examples of exact cover problems include Sudoku and the N-Queens problem which we will see later, but first let's consider a 2X2 Latin Square to help illustrate what is meant by an exact cover problem.

1.2 Latin Square Problem

	0	1	2
0	A	B	C
1	B	C	A
2	C	A	B

A Latin Square is a puzzle similar to Sudoku, where an NXN grid must be filled with N distinct shapes, such that no two identical shapes share the same column or row.
Here we include notation that denotes where each symbol A,B,C lie, i.e. in above 3X3 example, A lies at $(0,0)$ and $(1,2)$ and $(2,1)$.
As an example we will construct a simple 2X2 Latin Square, posing the puzzle as an Exact Cover Problem.

	0	1
0		
1		

To pose this question in an exact cover problem context, let's break down the **constraints** of the 2X2 puzzle, where C_i refers to the numbering of the constraint.

First, we consider the fact that each square in the grid needs a **symbol**:
C_1 : (0,0) must have a symbol
C_2 : (0,1) must have a symbol
C_3 : (1,0) must have a symbol
C_4 : (1,1) must have a symbol
Next we consider the **row constraint**:
C_5 : Symbol A must appear in row 1
C_6 : Symbol B must appear in row 1
C_7 : Symbol A must appear in row 2
C_8 : Symbol B must appear in row 2
Finally we consider the **column constraint**,
C_9 : Symbol A must appear in column 1
C_{10} : Symbol B must appear in column 1
C_{11} : Symbol A must appear in column 2
C_{12} : Symbol B must appear in column 2
These are our 12 constraints for the puzzle. Now we can construct a checklist to that might help us solve the puzzle, where C_i are the constraints, and S@(x,y) refers to the symbol at coordinates (x,y)

	Constraints											
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}
$A@(0,0)$	✓	X	X	X	✓	X	X	X	✓	X	X	X
$A@(0,1)$	X	✓	X	X	X	✓	X	X	✓	X	X	X
$A@(1,0)$	X	X	✓	X	✓	X	X	X	X	✓	X	X
$A@(1,1)$	X	X	X	✓	X	✓	X	X	X	✓	X	X
$B@(0,0)$	✓	X	X	X	X	X	✓	X	X	X	✓	X
$B@(0,1)$	X	✓	X	X	X	X	X	✓	X	X	✓	X
$B@(1,0)$	X	X	✓	X	X	X	✓	X	X	X	X	✓
$B@(1,1)$	X	X	X	✓	X	X	X	✓	X	X	X	✓

Since we have our checklist with all necessary constraints and placements, we can directly construct a "One-Zero" matrix by simply changing ✓ to ones and X to zeros while retaining the $8X12$ shape of the table, which we will denote as A for this example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now that we have our problem posed in a suitable format, let's venture into our goal, as Donald Knuth puts it:
"Given a matrix of 0s and 1s, does it have a set of rows containing exactly one 1 in each column?" [1]

1.3 Solution to the Latin Square

Deducing a solution using trial and error in this case is quite trivial since there are so few options, even without constructing A one might deduce the following solution:

	0	1
0	A	B
1	B	A

This gives us one possible solution with the following choices; A@(0,0), B@(0,1), B@(1,0), and A@(1,1). We can eliminate the other options in the table:

	Constraints											
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}
$A@(0,0)$	✓	X	X	X	✓	X	X	X	✓	X	X	X
$A@(1,1)$	X	X	X	✓	X	✓	X	X	X	✓	X	X
$B@(0,1)$	X	✓	X	X	X	X	X	✓	X	X	✓	X
$B@(1,0)$	X	X	✓	X	X	X	✓	X	X	X	X	✓

Here we see the solution to the problem, where every constraint is filled exactly once. We can denote in matrix form as:

$$A^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In A^* we see exactly one 1 in every column. This satisfies Knuth's criteria for a solution to the Exact Cover Method but our method is not reliable. This was a very simple example where a trial and error approach gives us the answer quickly but in cases such as the N-Queens problem, for large N we have a much larger number of both choices (rows) at N^2 and constraints (columns) at $6*(N-1)$. In these less trivial cases the Exact Cover Problem is considered **NP-Complete**.

2 Algorithm X and Dancing Links

2.1 NP-Complete

[...]

2.2 Algorithm X

Algorithm X is a **non-deterministic**, **recursive**, **backtracking**, **depth-first** algorithm used to finds all solutions to the exact cover problem defined by any given matrix A of 0s and 1s.
Non-deterministic algorithms, given a particular input, will not always produce the same output. This is essential for an exact cover problem as the algorithm isn't restricted to finding only one solution.
Recursive algorithms will rerun subroutines and functions until a specific condition is met, allowing efficiency by reusing a block of code until a solution is found.
Backtracking is [...]
Depth-first refers to [...]

2.3 Dancing Links

3 The N-Queens Problem

3.1 Forming the 1-0 Matrix

This general approach will be the first step toward solving the N-Queens problem. This matrix will be called the 1-0 matrix henceforth. The 1-0 matrix has a particular structure in the context of this problem, namely: The columns correspond to the satisfaction of the problem's constraints, while the rows correspond to all the possible queen placements. For each placement, there will be a 1 for each satisfied constraint, and a 0 otherwise. Details of the constraints are discussed below, but first we must define the board and coordinates clearly. In standard chess we refer to horizontal rows of the board as ranks, and vertical columns as files. Although in chess it is standard to begin counting at 1, in this context we will begin at 0 (We are using python after all). Here we can see the ranks and files of a size n=4 chessboard:

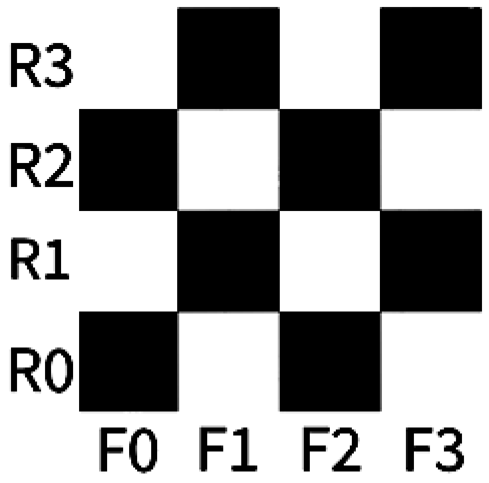


Figure 1: The Rank and File constraints

Now we must discuss how this matrix is formed for a board of size n.

Speaking technically, we will not solve the actual problem using a matrix or NumPy array, instead we will create a set of doubly linked circular lists, running vertically and horizontally. However, it seems the most direct way to create this linked list, is to first create a NumPy array, with the 1s and 0s correctly filled, and then to convert it into a doubly linked circular list.

The columns of the 1-0 matrix correspond to the various constraints in the problem, namely two primary and two secondary constraints: Primary 1.) There must be one and only one queen per rank. 2.) There must be one and only one queen per file. Secondary 3.) There can be at most one queen per diagonal. 4.) There can be at most one queen per reverse-diagonal. The rows of this matrix correspond to each of the possible queen placements, for example a queen at (0,0) or at (2,3).

3.2 Numbering the constraints

It is worth noting here that we can disregard the four diagonals of length one, namely the two diagonals at points (0, n) and (n, 0) and the two reverse-diagonals at (0, 0) and (n, n). If a queen is occupying a diagonal of length one, it is simply not possible to have another queen occupy the same diagonal, therefore there is no need to 'record' whether or not such a constraint has been satisfied.

There are exactly 2n – 3 diagonals (and reverse-diagonals) on the board.

Consider the uppermost rank on the board and the first file (the left and top edges), each square has one diagonal originating from it, and traversing the board toward the bottom right. This would give us a total of 2n diagonals, n for each side. We must subtract two for the aforementioned diagonals of length one, which originate in the corners, and also subtract an additional one for the 'middle' diagonal, which will be counted twice. This logic shows there are 2n-3 diagonals (and similarly 2n-3 reverse-diagonals).

It is helpful to number these constraints, as these numeric labels will correspond to the constraints (the columns) of the 1-0 matrix. The rank constraints will be numbered 0,1, . . . , n-1. The file constraints will be numbered n, n+1, . . . , 2n-1. The diagonal constraints will be numbered 2n, 2n+1, . . . , 4n-4. The reverse-diagonal constraints will be numbered 4n-3, 4n-3, . . . , 6n-7.

Below we can see a simple diagram illustrating these constraints for a n=4 chessboard. The diagonals are coloured in blue, while the back diagonals are coloured in red.

Now we must discuss how this matrix is formed for a board of size n.

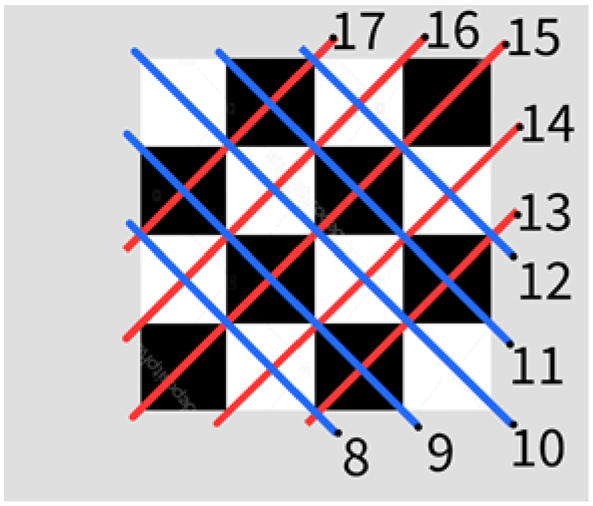


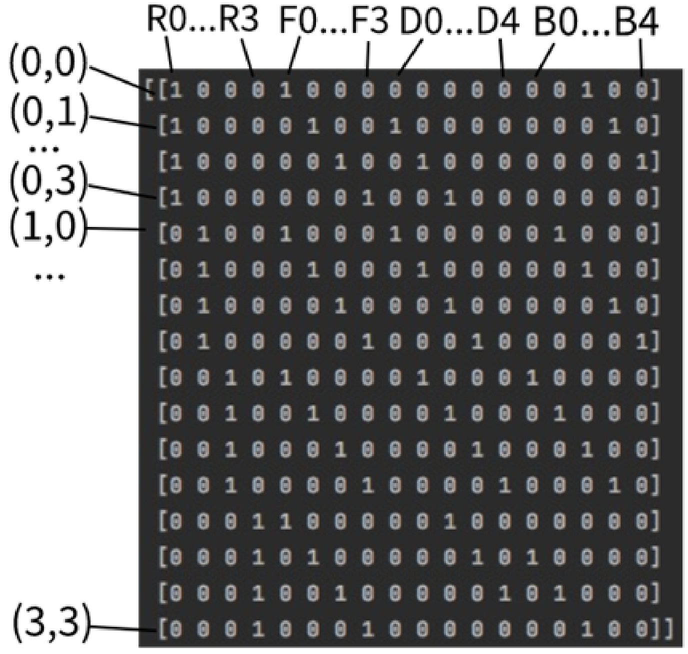
Figure 2: The Diagonal (Blue) and Reverse-Diagonal (Red) constraints

3.3 A method for calculating the constraints

Now we can calculate all the constraints which are satisfied by the various positions of queens:

$$(i, j) \forall i \geq 0, j \leq n - 1$$

We can see the python output for the case n=4 here, with the constraints roughly labelled following the outline specified before; ranks 0 through n-1, files 0 through n-1 then diagonals and reverse-diagonals. Along the rows we can see the various queen placements, first (0,0) through (0, n-1), then (1,0) through (1, n-1) and so on.



```
[[1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0]
 [0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0]
 [0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1]
 [0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0]
 [0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0]
 [0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0]
 [0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0]
 [0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0]
 [0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0]]
```

Figure 3: Labelled console output of the 1-0 matrix

3.4 Conversion to a doubly linked list

Three separate classes are used for the formation of this list, first a CircularList class, next a column class and lastly a node class.

The node class simply contains four attributes: up, down, left and right. These point to the other nodes around this node and will be used in the DLX algorithm to cover and uncover rows. The column class contains the same four attributes as the node class, as well as two additional attributes: name and size. Name is a string which will allow humans to interpret which constraint this column represents (Rank 0 or Diagonal 3 for example), size is an integer which refers to the number of nodes or 1s, that are below a particular column header, and is used to select optimal rows during DLX. The CircularList class contains only one attribute, a column object with the name ‘Master’, the entire list object will be built around this, and during DLX it will serve as a starting pointer to traverse the list. This master node lives left of the first column header, and as such the up, down and size attributes are not used.

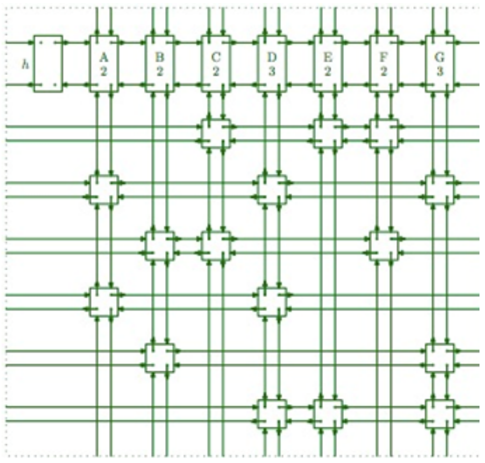


Figure 4: Illustration of a doubly linked circular list from Donald Knuth’s Dancing Links paper

```
A class specific function convert_one_zero , is used to convert the 1-0 matrix into a circul
```

```
Briefly in pseudocode:

Repeat for each column of 1-0 matrix:
Create column node
Name column node
Connect column node to the previous column node
Repeat for each row of 1-0 matrix:
Search the row
If row[i] is 1:
```

```
Create a node
If not first node in row:
Connect node.left to the previous node
Find the corresponding column header
Search until the bottom of this column/vertical list is found
Connect below to node
Repeat for each column:
Traverse column to the bottom
Update column.size accordingly
Connect ‘lowest’ node.down to the header
Connect header.up to the ‘lowest’ node
Connect furthest right column header.right to the master
Connect master.left to the furthest right column header

This function converts the 1-0 matrix into a doubly linked circular list.
```

4 Collaboration Outline

[not done]
Github, slack where we outlined collaborative goals, priorities, informal deadlines, weekly zoom meetings, work journal ...

References

[1] D. Knuth. Dancing links, 2000.

"' **This page not for final script**
To do list for Introduction:
define the following:
backtracking
depth-first
reduction transformations
NP-Complete

Fully introduce Algorithm x
find reduced solution to 2x2 and make it into a table
Define dancing links
draw diagram of dancing links arrows for 2x2 latin, covering
Format of manuscript: (From Blackboard)

- Cover page
- Declaration page
- Introduction
- Main body (structured into sections as per topic requires).
- Collaboration Outline
- Conclusion
- Bibliography
- Appendices (if applicable)

This page just for reference "'