Project Report on Numbrix

About Numbrix

Numbrix is a puzzle game played on an N x N grid. The numbers 1 to N^2 must be arranged in a continuous path that fits into the square grid. Consecutive numbers must be placed horizontally or vertically adjacent to each other.

Implementation of the game in LISP

The game has been implemented in LISP using Allegro CL 9.0. There are two modes in which the game can be played: manual and auto mode.

Manual Mode:
A user can play the game by choosing any board from among the various boards that have been hard-coded in the program. Once a board is chosen, missing values can be entered into the grid. Values entered previously can also be changed (except the ones which are initially present on the board). While writing a value on the board, the user must enter the row and column value of the grid in which the number is to be written. The program checks if the row value, column value and the number entered are inbound or not, i.e. the row and column numbers must lie between 1 and N, and the number must lie between 1 and N^2. If any of the three values are incorrect, an error is printed stating which value was out of bound. If all the values are entered correctly, then the program checks if the user is trying to alter a number initially present on the board. An error is thrown in such an event. If that is not the case, the number is written on the board, provided the number is not already present on the board. This process is repeated till there is no empty cell left on the board. As soon as the last number is written, the program checks if the values entered are in the correct positions or not. It first searches for '1' on the board. Starting from 1, it searches which of the adjacent cells (horizontal and vertical) has the number 2. If it finds 2, it looks for 3 in cells adjacent to 2, and so on. This process is repeated till all numbers are traversed. However, if at any point the next number is not found in any of the adjacent cells, then the board has been filled incorrectly and the game stops. If all numbers till N^2 are traversed, then the board has been filled correctly.

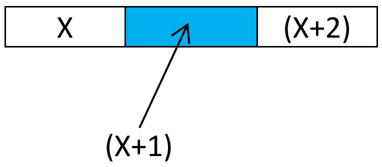
Auto Mode:
In the auto mode, the user selects a board and the program solves it by applying various heuristics. After applying these heuristics, the program displays the final board and traverses it from 1 to N^2 in the same manner as done in the manual mode. The heuristics applies are:

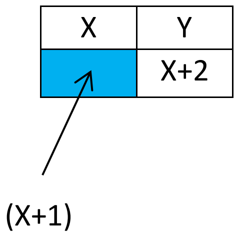
1. Move possible only in one direction

   Starting from 1 to N^2, the program looks for cells for which a move is possible only in one direction (up/down/left/right). If the number in that cell is X, then it searches for (X-1) and (X+1) on the board. If both the numbers are present, then no number is added to the cell adjacent to X. If (X+1) is not present on the board, then (X+1) added, otherwise (X-1) is added to the board adjacent to X.

2. Alternate numbers present

   Starting from 1 to N^2, the program searches for alternate numbers present on the board. Suppose it is on a cell with a number X in it. It first searches if (X+1) is present on the board or not. If (X+1) is not present, then it searches for (X+2) on the board. If (X+2) is present, then the program checks if X and (X+2) lie in the same row or column. If they do, (X+1) is placed in the cell between them. If X and (X+2) lie in cells diagonal to each other, then it checks if the two cells which are adjacent to both X and (X+2) are empty or not. If both only one of the cells is empty, then (X+1) is placed in that cell.





3. Moves possible in two directions

	22		
20	21	36	
	🟦	37	38
	X	(X-1)	
	56	57	

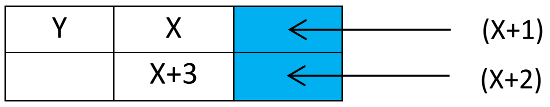
   The program searches cells for which a move is possible in two directions. Let the program be at the cell with the number X in it. It first checks if its predecessor and successor are present on the board or not. If only one of them is present, then it checks which two adjacent cells are empty.

   For example, in the board above, for the cell X, the cell above it and the one to its left are empty. The program searches for (X-1) and (X+1). It finds that (X-1) is present. Considering the cell above X, the left cell is empty. The right cell has a number, whose predecessor and successor are already on the board. Similar is the case for the cell above it. If such is the situation, then (X+1) is placed in the cell above X.

The three heuristics stated above are applied until none of them can fill anymore cells. Next, the program tries to look for some patterns on the board:

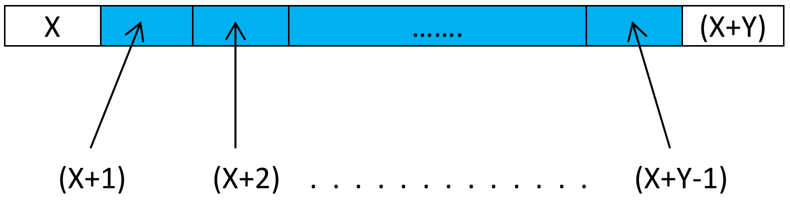
4. Starting from 1 to N^2, the program searches for any two numbers which have a gap of 2 or more numbers between them, i.e. 2 or more consecutive numbers are missing from the board between a pair of numbers already present.

If two numbers are missing between a pair X and (X+3), and the pair of numbers are adjacent to each other, then



This can be done only if, at least one of the cells to the left of x and (x+3) is filled.

If the pair is X and (X+Y), then the numbers missing between them are (Y-1). If X and (X+Y) are in the same row or column, and they have exactly (Y-1) cells between them, then, (X+1), (X+2), …, (X+Y-1) are placed in the cells between X and (X+Y).



Heuristics 1 to 4 are repeated till none of them can fill anymore cells.

5. Finally, the program performs iterative deepening to fill the remaining empty cells, if any.

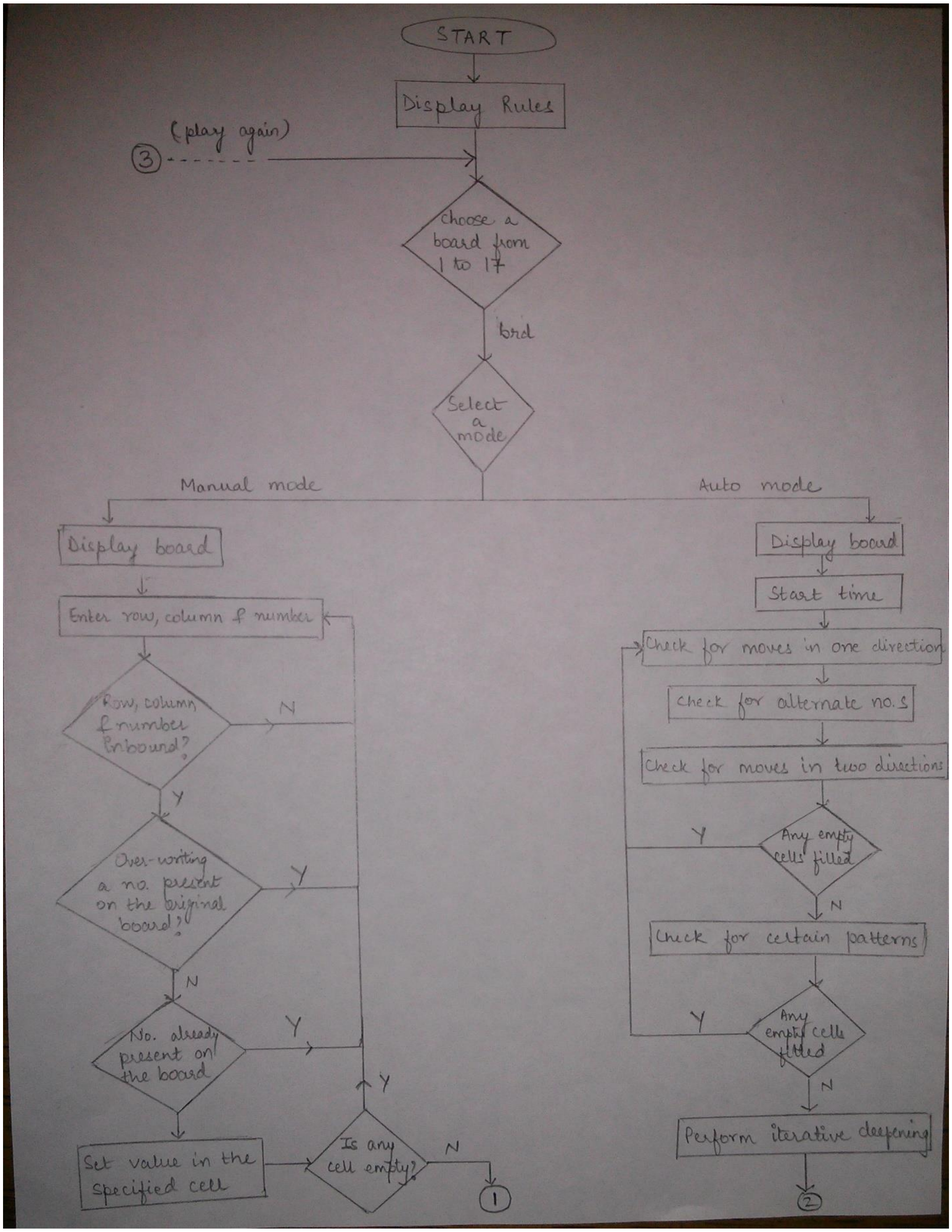
Main Variables

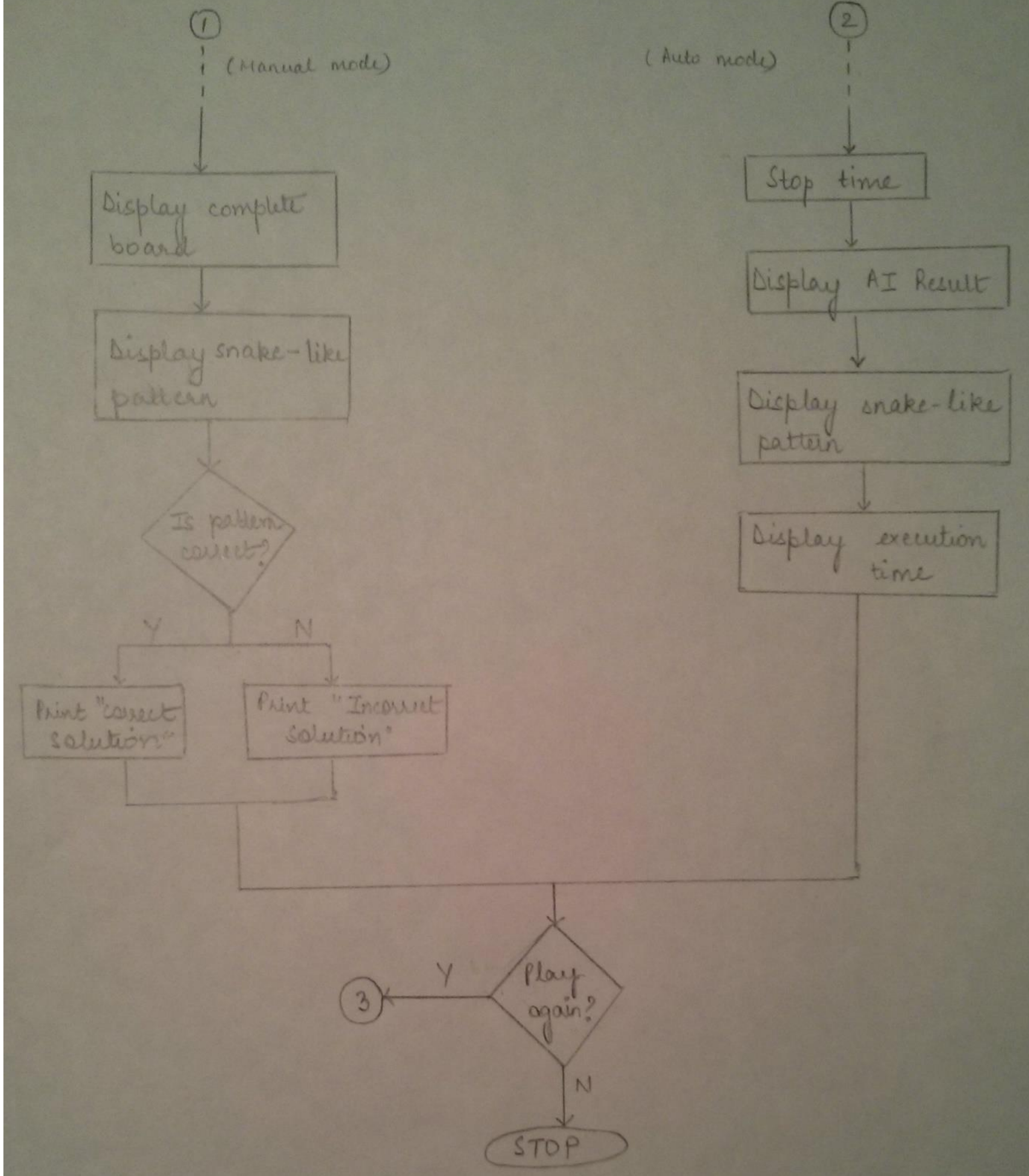
- brd – a list of lists representing the board chosen to play Numbrix.
- empty-brd – a list of lists representing an empty board for the board being filled, to show the traversing from 1 to N^2 (snake-like pattern).
- available – a list containing 0's and 1's. The first position in the list corresponds to the number '1' on the board, the second number corresponds to the number '2' on the board and so on. If a number is present, its position in 'available' will be 0, else it will be 1.
- flag – a list containing all 0's initially. Used for backtracking in iterative deepening.
- counter1, counter2, counter3 and counter4 – counters to check when to start iterative deepening.

Main Functions/Methods

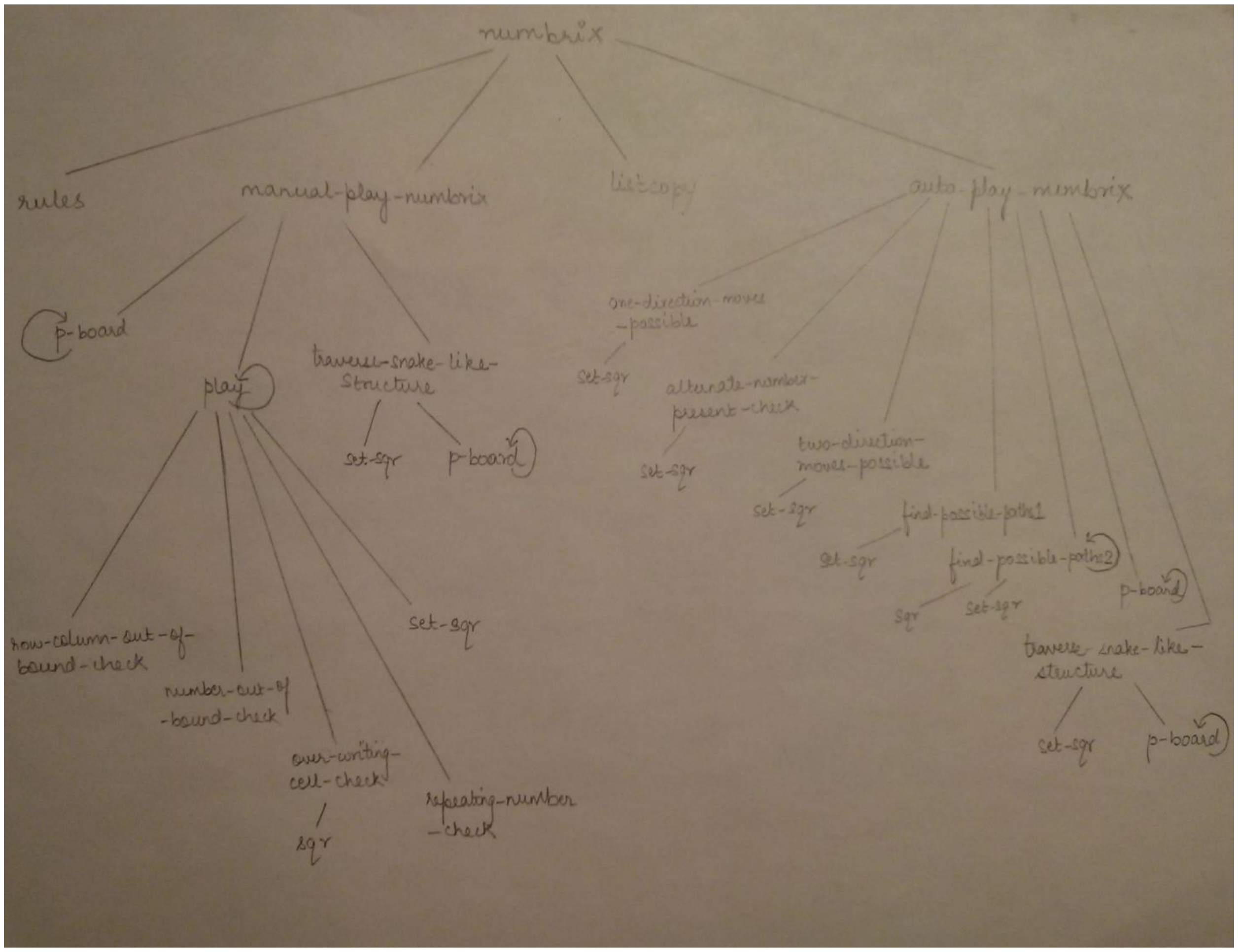
- numbrix – starts the game and asks the user to select the mode and a board to play
- rules – displays the rules of the game
- listcopy – makes a copy of the original board
- auto-play-numbrix – game played in the auto mode, i.e., by the program
- one-direction-moves-possible – checks if moves possible only in one direction for a particular cell (heuristic 1 of auto mode)
- alternate-numbers-present-check – checks if X and (X+2) are present and (X+1) is missing or not (heuristic 2 of auto mode)
- two-direction-moves-possible – checks if moves are possible in two direction for a particular cell (heuristic 3 of auto mode)
- find-possible-paths1 – searches for certain patterns on the board (heuristic 4 of auto mode)
- find-possible-paths2 – performs iterative deepening
- manual-play-numbrix – game played in the manual mode, i.e., by the user
- row-column-out-of-bound-check – checks if row and column number entered by the user are out of bound
- number-out-of-bound-check – checks if number entered by the user is out of bound
- over-writing-cell-check – checks if the number being over-written is part of the original board
- repeating-number-check – checks if the number being written is already present on the board
- traverse-snake-like-structure – traverses the board from 1 to N^2
- sqr – finds the value present for a given row and column number
- set-sqr – sets a given value to the specified cell
- p-board – prints the board

Flow Chart


START
Display Rules
(play again)
3
Choose a board from 1 to 17
brd
Select a mode
Manual mode
Auto mode
Display board
Enter row, column & number
Row, column & number inbound?
N
Y
Over-writing a no. present on the original board?
Y
N
No. already present on the board
Y
Set value in the specified cell
Is any cell empty?
Y
N
1
Display board
Start time
Check for moves in one direction
Check for alternate no.s
Check for moves in two directions
Any empty cells filled
Y
N
Check for certain patterns
Any empty cells filled
Y
N
Perform iterative deepening
2


1
(Manual mode)
2
(Auto mode)
Display complete board
Display snake-like pattern
Is pattern correct?
Y
N
Print "correct solution"
Print "Incorrect solution"
Stop time
Display AI Result
Display snake-like pattern
Display execution time
Play again?
Y
3
N
STOP

Calling Hierarchy



‘numbrix’ is the main function. First it calls ‘rules’ and displays the rules. Then it makes a copy of the board chosen using ‘listcopy’. Next, it either calls ‘manual-play-numbrix’ or ‘auto-play-numbrix’ depending on the users choice.

‘manual-play-numbrix’ first displays the board by ‘p-board’. It calls a function ‘play’ which allows the user to input numbers to the board. ‘play’ calls itself recursively until no empty space is left on the board. Finally, ‘traverse-snake-like-structure’ checks if the values entered are correct or not by traversing the board in a snake-like pattern.

‘auto-play-numbrix’ checks every heuristic till the board is filled completely. It prints the final board by ‘p-board’ and then traverses the board by ‘traverse-snake-like-structure’.

‘numbrix’ uses a loop to determine if the user wants to play the game again or not.

Intelligence implemented

- Move possible only in one direction
- Alternate numbers present in diagonal cells or same row or column
- Move possible in two directions
- Searching for certain patterns on the board
- Iterative deepening and backtracking

Intelligence not implemented

- Filling numbers along the virtual boundary (blue cells) created when none of the heuristics apply (except iterative deepening)

45	44	41	40	37	36	7	6	5
46	43	42	39	38	35	8	1	4
47	48	49	🟦	🟦	34	9	2	3
52	51	50	🟦		🟦	10	11	12
53	🟦	🟦				🟦	🟦	13
🟦							🟦	14
75	🟦					🟦	16	15
76	79	🟦	🟦	🟦		🟦	23	24
77	78	67	66	65	🟦	27	26	25

This heuristic has not been implemented since it was proving very difficult to form a generalized solution. Instead, iterative deepening has been used.

What I would do differently if I could start over

Since iterative deepening is not an efficient way to find a solution, I would try to come up with better heuristics to solve the problem when numbers missing between a pair of numbers is greater than 2. I would also want to reduce the amount of backtracking required.