Report of TCG Homework 1

How to compile/run the code

Use python *.py [n] [filename] to run the code

The file should gave one puzzle a time, as the example.txt shows in the example_input folder.

Algorithm and heuristic I've implemented

1. BFS.py //use BFS to fill in row/column combination and search thru the whole tree

2. DFS.py //using check_board function in every DFS_find_solution to eliminate the possible of the column and row combination with (1) fill in the board with column constraint and find out the cell that would definitely be black or white, and eliminate the row possible combination with the board. (2) fill in the board with row constraint and find out the cell that would definitely be black or white, and eliminate the column possible combination with the board. (3) keep doing step1 and step2 until no change in row/column combination.

Basically only check_board heuristic had been implemented. Although there are several case of logic ad-hoc heuristic[1], they can be interpret as the finding the intersection of all combinations, to make good use of the information we found and apply all the procedures, we eliminate the impossible row/column combination after finding the intersect of column combination and vice versa.

The comparison of searching algorithm

Following table shows the speed of BFS and DFS performed on 15x15 nonogram.

15 x 15	$1	$2	$3	$4	$5	$6	$7	$8	$9	$10
BFS	> 15mins									
DFS	0.162s	0.216s	1.857s	0.45s	30.29s	1.352s	14.12s	0.982s	20.15s	30.29s

BFS has to search the board in each leaf node sequentially and compare to the column/row constrain. This makes it take a large amount of time to find the correct answer.

Although DFS with heuristic function takes a huge amount of memory storing the current board and current combination in every recursive function, it is much more faster than BFS.

I've also found my DFS sometimes output differently from the provide answer, so i print row/column information of the current solution and found it's exactly the same to the

[1] S. Salcedo-Sanz, E.G. Ortiz-Garcia, A.M. Perez-Bellido, J.A. Portilla-Figueras, and X. Yao. Solving Japanese puzzles with heuristics. In Proceed-ings IEEE Symposium on Computational Intelligence and Games (CIG),pages 224{231, 2007.

constrains we gave. Thus, there might be several solutions for one problems and my algorithm are able to find out all of them, but I only print out one in the output.

The game complexity analysis

We can see Nonograms as finding either a box is 0 or 1 for the whole board. The rows and columns form 2*n n-SAT problem which is NP-complete (proof is omitted). The game complexity is $2^{n\char`^2}$.

However, from the clues we can find a series of combination for each row and column, which lower the game complexity to around b^n, where b is the maximum number of combination for a column or row.

The factors affect the performance of each algorithm

The factors the mainly affect the performance of BFS and DFS is the number of branches and depth. In nonogram the depth is fixed to the board size, both BFS and DFS would take a longer time if the combination of row/columns are a lot, which indicated a lot of branches.

Another factor that affect the performance of DFS with heuristic function is how much combination we can eliminate after doing pre-process. Those with high elimination rate of combinations give a better performance.

The last factor is how early can we prune the branch, in other words, which is based on the case of first few columns of the board. This highly affect performance of DFS, as it tries to prune as much branch as possible.

The factors affect the difficulty of Nonogram

The main factor that affect the difficulty of Nonogram is the problem complexity which is the size of n, which also represents the tree height of the DFS/BFS search tree.

Another huge factor is about the combinations. We can observe that some clues give a very limited number of possible combinations but some give a lot, especially those that is medium sparse. Noted in here we consider the space after black cells comes with a fixed white cell after it, the number of combinations is to fill white spaces in between and at the begin or end of these blocks, which is maximum in between 1/3 to 1/2 depends on size of each block.

[1] S. Salcedo-Sanz, E.G. Ortiz-Garcia, A.M. Perez-Bellido, J.A. Portilla-Figueras, and X. Yao. Solving Japanese puzzles with heuristics. In Proceed-ings IEEE Symposium on Computational Intelligence and Games (CIG),pages 224{231, 2007.

Other observation or discussion

Nonograms is a n-SAT problem with clues! Something that most people include me had not notice before.

A lot of nonograms problem with certain pattern can use the trick of guessing its column is symmetric, as a lot of pattern have some columns that is symmetric. *Not applicable on random generated problem

Turning the combination set into binary is good, as the overlapping can be found by using OR, AND and NOT operation, however the eliminating process is harder to code. My binary code end up with lots of bugs that I have difficulty tracking it. Moreover, since the board is easier to be check with double dimensional array and overlapping process only takes about few seconds, the speed up may not be that essential.

[1] S. Salcedo-Sanz, E.G. Ortiz-Garcia, A.M. Perez-Bellido, J.A. Portilla-Figueras, and X. Yao. Solving Japanese puzzles with heuristics. In Proceed-ings IEEE Symposium on Computational Intelligence and Games (CIG),pages 224{231, 2007.