Lab 1

Manu Patil, Evan New-Schmidt

March 2020

1 Introduction

For our game exploration, we're looking at nonogram puzzles.

2 Proving NP-Completeness

We offer a reduction from the bin packing problem to nonograms. Specifically, reducing the validation of a bin-packing solution to the validation of a nonogram.

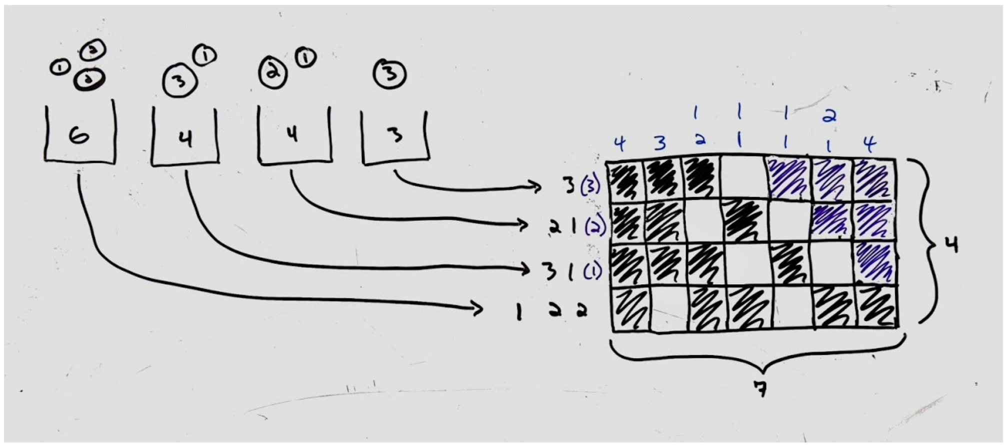


Figure 1: **Reduction from Bin Packing Solution to Nonogram**. The bins with assigned items (left) are converted to a nonogram (right), with dummy runs in purple to account for bin sizes and determined vertical runs in blue.

Consider a bin packing solution with N differently-sized bins with assigned items, also of different size. The bin packing solution is valid if all items fit into their assigned bins. To reduce this solution to a nonogram, convert each bin into a row in a nonogram, and convert the items assigned to the bin to runs in the appropriate row. Because nonogram rows require at least one space between runs, the width of the nonogram must accommodate the size of the largest bin and spacing for all of the items, expressed as

$$w = \max_n(b_n + |I_n| - 1)$$

where b_n is the size of bin n and I_n is the set of items in bin n. The height h of the resulting nonogram is the same as the number of bins

$$h = N$$

Because the bins are all different sizes, dummy runs are added to all bin-rows that are smaller than w, such that

$$d_n = \max(0, w - (b_n + |I_n| - 1) - 1)$$

where d_n is the length of a dummy run.

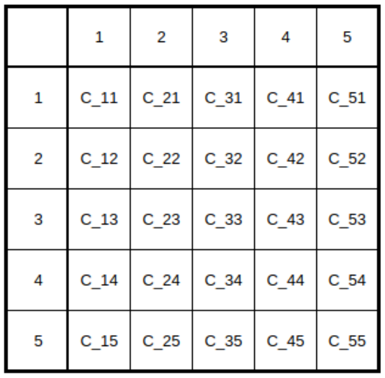
Looking only at the rows of this nonogram, the nonogram is valid if and only if the item-runs can fit within the bin-rows. While the item-runs are ordered as compared to the unordered bin packing version, because there are no column constraints the order does not affect whether or not the item-runs will fit in the bin-rows.

While the nonogram in it's current state fully captures the validity of the bin packing solution, standard nonograms require runs for columns as well. To complete this, column runs are added that describe the layout caused by left-aligning all of the row runs. How the row-runs are aligned does not affect whether or not they will successfully fit, so these added column constraints do not affect the validity of the nonogram.

If the constructed nonogram is valid, then the bin-packing solution is also valid, and if the constructed nonogram is invalid, then the bin-packing solution is also invalid.

3 Solver Implementation

Our solver implementation works in two stages. The first is converting the nonogram problem to a SAT problem and the second is to then use a SAT solver to find a solution
The conversion from nonograms to SAT is as follows. First, label each cell in the following format C_ij where i corresponds to row and j to column as seen in Figure 2



	1	2	3	4	5
1	C_11	C_21	C_31	C_41	C_51
2	C_12	C_22	C_32	C_42	C_52
3	C_13	C_23	C_33	C_43	C_53
4	C_14	C_24	C_34	C_44	C_54
5	C_15	C_25	C_35	C_45	C_55

Figure 2: Cell Naming Scheme

From here we recursively create all the row possibilities. For a row of length 10 and runs [3, 2, 2], the generated possibilities would include those shown in Figure 3

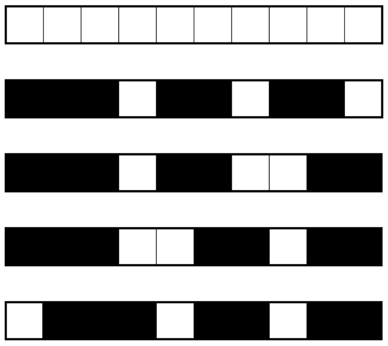


Figure 3: Possible Runs for length 10 and runs [3, 2, 2]

With these possibilities created we proceed to convert our problem to a CNF. Lets define some variables to make this a bit cleaner.
R = # of Rows
C = # of Columns
P_n = # of possibilities for row or column n
Our pseudo-CNF took the following form:

$$(\bigwedge_{j=1}^{R}\bigvee_{p=1}^{P_j}\bigwedge_{i=1}^{C} C_{ij})\bigwedge(\bigwedge_{i=1}^{C}\bigvee_{p=1}^{P_i}\bigwedge_{j=1}^{R} C_{ij}) \tag{1}$$

To build Equation 1 up. The innermost conjunction operator puts an **and** in between each of the cells in a possibility. Next, the the disjunction operates on all the possibilities in a row or column. Then we conjunct the result from the previous step to get something that represents all row or column configurations. Finally, we conjunct the row and column outputs to represent the entire grid.
In order to get this into formal CNF we used a built-in function of the sympy library which uses some clever tricks to move around the nested conjunction operator. The performance of this function seems to be prohibitive for nonogram problems of size $n \geq 8$, but there may be a more performant solution that could be optimized with future work.
This output is then plugged into a sat solver.
We later discovered the Python EDA library which is another library the offers Symbolic Boolean algebra operations and a SAT solver as well. In addition this library provided several encodings to make SAT solving more efficient. The one we chose was the Tseitin Encoding which significantly the speed of our solver and allowed for us to run the latter examples.

4 Test Cases

Below are a couple of our smaller test cases:

```
. # # .  2
# # . #  2 1
2 1 1 1
```

```
. . # # #  3
. . # # #  3
. . . # .  1
# # . . .  2
# . # # #  1 3
2 1 2 3 2
    1 1 1
```

```
. # # # . . . .  3
# # . # . . . .  2 1
. # # # . . # #  3 2
. . # # . . # #  2 2
. . # # # # # #  6
# . # # # # # .  1 5
# # # # # # . .  6
. . . . # . . .  1
. . . # # . . .  2
1 3 1 7 5 3 4 3
2 1 5 1
```

4.1 Test Cases enabled by Python EDA solver

```
. . . . . . . . . . # # # # # # . . . .  6
. . . . . . . . # # # . # . . # # # . .  3 1 3
. . . # . . # # # . . . # . . . . # # #  1 3 1 3
. . # # # . # # # # # # # # # # # # # #  3 14
. . . # . . # . . . . . . . . . . . . #  1 1 1
. . # . # . # # . . . . . . . . . . # #  1 1 2 2
# # # # # . . # # . . . . . . . . # # .  5 2 2
# # # # # . . . # . . . . . . . . # . .  5 1 1
# # # # # . . # # # . # # # . # # # . .  5 3 3 3
# # # # # # # # # . # # # . # # # . # # #  8 3 3 3
4 4 1 3 1 1 4 2 3 1 2 1 4 1 1 2 1 3 2 4
    5 4 5   1 2 3 1 1 1 1 1 1 1 4 2 1
              2   2 1 2   2 1 2 1   1
```

```
. . . . . . . . . . . . . . . . . . . . # # # # #  5
. . # # . . . . . . . . . . . . . . # # # . . # #  2 3 2
. # # . . . . . . . . . . . . . . # # # # # . . #  2 5 1
# # . . . . . . . . . . . . . # # # # # # # # . .  2 8
# # . . . . # # # # # . # # # # # # # # # # # . .  2 5 11
# . # . . # # . . . . # . . . . # # # # # # . . .  1 1 2 1 6
# . . # # . . . . . # . . . . . . . # # # . . . .  1 2 1 3
# # . . . . . . . . # . . . . . . . . . . . . . #  2 1 1
. # # . . . . . # # # # # # . . . . . . . . . # #  2 6 2
. . # # # # # # # # # # # # # # # . . . . # # # #  15 4
. . . . . # # # # # # # # # # . . # # # # # # # #  10 8
. . . . # # . # . # # # # . # # # . . # # # # # #  2 1 4 3 6
. . . . . . . . # # # # # # # # # # # # # # # # #  17
. . . . . . . . # # # # # # # # # # # # # # # # #  17
. . . . . . . # # # # # # # # # # # # # # # # # #  18
. . . . . . . # . . . # # # # # # # # # # # # # #  1 14
. . . . . . . # . # . # # # # # # # # # # # # # #  1 1 14
. . . . . . . . # # # # # . . . # # # # # # # # #  5 9
. . . . . . . . . . . . . . . . . # # # # # # # #  8
. . . . . . . . . . . . . . . . . . # # # # # # #  7
5 3 2 1 1 1 2 1 1 1 1 1 1 1 1 2 3 4 6 6 7 1 1 2 3
  2 1 1 1 3 2 3 3 7 9 10 10 3 8 1 1 1 1 10 10 4 2 12 13
    2 1 1     3 3 2 1     5   6 7 7 8     11 11
                1
```

```
. . . . # # # . # . . . . . . . . . . .  3 1
. . . . # # . # # # # . # . . . . . . .  2 4 1
. . . . # . # # # . # # # . . . . . . .  1 3 3
. . # # . # # # # . . . . . . . . . . .  2 4
. # # # . # # # . # . . . . # # # . . .  3 3 1 3
# # # . . # # . # # . . . # . # # # . .  3 2 2 1 3
# # . . # # . # # . . . . # # . # # . .  2 2 2 2 2
. . . . # # . # . # . . # # . # . # . .  2 1 1 2 1 1
. . . . # . # # . # . . . # # # # . . .  1 2 1 4
. . . . # . # . # # . . . . . # # . . .  1 1 2 2
. . . . . # # . # # . . # # # # # # # #  2 2 8
. . . . # # . # # . . . # # . . # # # #  2 2 2 4
. . . . # . # # . # # . # . . . # . . #  1 2 2 1 1 1
# # # . . # # # . # # # # # . . . . . #  3 3 5 1
# . # . # # # . # . . . . # . . . . # #  1 1 3 1 1 2
# # . . # # # . # . . . . # # # . # # #  2 3 1 3 3
. # . # # # . # # . # # # # # # # # . .  1 3 2 8
. # # # # . # # # . # # # # # # # # . .  4 3 8
. . . # . # # # # . # # . # # # # # . .  1 4 2 5
. . . # . # # # # . # # . . . # # . . .  1 4 2 2
. . . . # # # # . . # # . . . # # # # #  4 2 5
. . . # # # # # . # # # . . . # # # # #  5 3 5
. . . # # # # . # . . . . . . . . . . #  4 1 1
. . # # # # . # # . . . . . . . . . . .  4 2
. . # # # . # # # . . . . . . . . . . .  3 3
2 3 3 2 3 2 1 4 4 1 2 1 2 4 1 2 3 3 2 6
3 1 2 4 4 5 4 3 2 2 2 1 1 2 1 4 5 2 2 3
  3 1 4 2 2 3 3 3 4 6 6 4 6 1 7 6 4 2
    2   4 4 4 6 6 2     2   1     2
        5 6 6 2 3 1         4
            1
```