A COMPRESSIVE SENSING APPROACH TO SOLVING NONOGRAMS

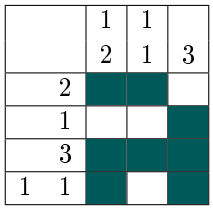
ISAAC J. LEE AND ÓSCAR LÓPEZ

Abstract. A nonogram is a logic puzzle where one shades certain cells of a 2D grid to reveal a hidden image. One uses the sequences of numbers on the left and the top of the grid to figure out how many and which cells to shade. We propose a new technique to solve a nonogram using compressive sensing. Our method avoids (1) partial fill-ins, (2) heuristics, and (3) over-complication, and only requires that we solve a binary integer programming problem. We sincerely hope that the simplicity in our approach will motivate fellow students to explore the area of compressive sensing.

Keywords: nonogram, compressive sensing, binary integer programming

1. Introduction

A nonogram is a logic puzzle where one shades certain cells of a 2D grid to reveal a hidden image. One uses the sequences of numbers on the left and the top of the grid to figure out how many and which cells to shade. Here is a completed nonogram that reveals a very stylish λ.



A sequence $(b_1, \cdots, b_p)$ means, we shade p blocks of $b_1, \cdots, b_p$ many cells, with at least one empty cell between two consecutive blocks. There may or may not be empty cells before the block of b_1 cells and after that of b_p cells.

Typically, one (a person or a computer program) uses logic to solve a nonogram. For a large grid size, this may be laborious (albeit deliciously challenging). In fact, determining if the solution to a nonogram exists, even with an assumption such as the solution being convex or connected, and determining if the solution is unique are both NP-complete problems [6, 7, 5]. Note, checking if we have the right solution can be done easily, in finite time.

In this paper, we solely focus on solving a nonogram. We will assume that the creator of the nonogram designed it such that the solution exists and is unique. These are very reasonable assumptions, since no one would solve a nonogram for which the solution does not exist or there exist multiple solutions. Other than that, the nonogram is as general as it can be. It has the size m-by-n, with m rows and n columns. For the **stylish lambda** above, we have $m = 4$ and $n = 3$.

People have developed different techniques to solve a nonogram; we bring up only a few here and refer to [8] for a comprehensive list. As previously mentioned, one typically uses logic to solve a nonogram. So some have tried to partially fill in cells using logic and heuristically decide which row or column to tackle next [2, 3]. They would apply some technique to the selected row or column, and repeat these steps until the nonogram is solved. More advanced (complicated) techniques include using an evolutionary algorithm [1, 4].

We briefly mention that a nonogram is related to a discrete tomography problem. In discrete tomography, we are interested in reconstructing an object from a number of its projections. Consider a two-dimensional object. We can make 2 one-dimensional projections—along the "row" direction and the "column" direction—and obtain the row sums r_i $(i = 1, \cdots, m)$ and the column sums s_j $(j = 1, \cdots, n)$. The question is, can we reconstruct the object from the values r_i and s_j?

Returning to nonograms, we propose a new technique using compressive sensing. Our method avoids (1) partial fill-ins, (2) heuristics, and (3) over-complication; it only requires that we solve a binary integer programming problem. Over the next three sections, we detail how to recast the solution to *any* nonogram as a sparse vector $\vec{x}$, and how to recover $\vec{x}$ from an l_1-minimization. We use the theory of compressive sensing to analyze our method. In the final two sections, we present a few nonograms we tried to solve, and comment on the results and on future work.

2. Complexity in Solving Nonograms

Solving nonograms is a NP-complete problem. However, proving so is neither trivial nor intuitive. In this section, we present a simple combinatorial argument that shows why an exhaustive search through the possible solutions is impractical. This warrants us to design more efficient methods—say, using compressive sensing—to find the solution. In addition, Lemma 2.1 and Theorem 2.3 will help us analyze our method in the next two sections.

Lemma 2.1 *Suppose the nonogram has been created randomly. In other words, in the creation process, we decided to shade each cell or leave it empty with an equal probability of $\frac{1}{2}$, independently of other cells.*

Then, for any row, the expected number of shaded cells, i.e. the expected row sum $\mathbb{E}(r)$, is equal to $\frac{n}{2}$. Furthermore, for any row, the expected number of blocks of shaded cells, $\mathbb{E}(p)$, is about $\frac{n}{4}$ (equal to $\frac{n+1}{4}$).

Proof. The first statement is clear: we can expect half of the cells on a row to be shaded, and the other half to be empty.

For the second, we appeal to the following problem: If we flip a fair coin n times, how many runs involving heads and tails can we expect? Let $X_i = 1$ if the i-th flip comes out heads, and $X_i = 0$ if tails.

For $i < n$, a run ends if and only if $X_i \neq X_{i+1}$ (occurs with probability $\frac{1}{2}$), whereas for the last flip $i = n$, a run always meets its end. Hence, the total number of runs, p, is equal to

$$p = \sum_{i=1}^{n-1} \mathbf{1}_{\{X_i \neq X_{i+1}\}}(X_i) + 1,$$

where $\mathbf{1}_{(\cdot)}$ is the indicator function. By the linearity of expectation, we get

$$\mathbb{E}(p) = \sum_{i=1}^{n-1} \mathbb{P}\big(X_i \neq X_{i+1}\big) + 1 = \frac{n-1}{2} + 1.$$

For the nonogram, we are interested only in the runs of heads. Take one-half of the expected number above. ∎

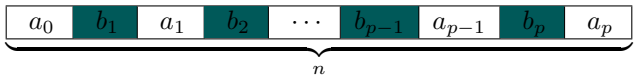
Lemma 2.2 *There are $\binom{n+p-1}{p-1}$ possible integer solutions to the system*

$$\begin{cases} a_1 + \cdots + a_p &= n \\ a_1, \cdots, a_p &\geq 0 \end{cases}.$$

Proof. We have n books to place on a shelf, and we also want to divide the books into p groups (a group may contain no books). Clearly, this requires that we have $(p-1)$ bookends. Hence, there will be $n + (p-1)$ objects on the shelf, and we will choose $(p-1)$ of them to be the bookends. ∎

Theorem 2.3 *Let $(b_1, \cdots, b_p)$ be a sequence given for one row of the nonogram, and let $r = b_1 + \cdots + b_p$ be the row sum. There are $\binom{n-r+1}{p}$ possible configurations that satisfy the sequence.*

Proof. Let $a_1, \cdots, a_{p-1}$ denote the number of empty cells between two consecutive blocks of shaded cells. Furthermore, let a_0 and a_p denote the number of empty cells before the first block and after the last block.



The numbers $b_1, \cdots, b_p$ are fixed, so it is the numbers $a_0, \cdots, a_p$ that dictate the possible configurations. The numbers $a_0, \cdots, a_p$ satisfy the system

$$\begin{cases} a_0 + a_1 + \cdots + a_{p-1} + a_p &= n - r \\ a_1, \cdots, a_{p-1} &\geq 1 \\ a_0, a_p &\geq 0 \end{cases},$$

or equivalently, with $a'_k = a_k - 1$ for $k = 1, \cdots, p-1$,

$$\begin{cases} a_0 + a'_1 + \cdots + a'_{p-1} + a_p &= n - r - (p-1) \\ a_0, a'_1, \cdots, a'_{p-1}, a_p &\geq 0 \end{cases}.$$

From Lemma 2.2, we know that there are $\binom{n-r+1}{p}$ possible solutions. ■

Corollary 2.4 *Let r_i and p_i denote the row sum and the number of blocks of shaded cells for the i-th row. An exhaustive search that takes into account the row sequences still requires that we check $\prod_{i=1}^{m} \binom{n-r_i+1}{p_i}$ number of configurations. Had the nonogram been created randomly, this number would be about $\binom{\frac{n}{2}+1}{\frac{n}{4}}^m$.*

Proof. The statement easily follows from Lemma 2.1 and Theorem 2.3. ■

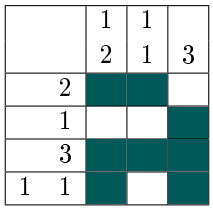
We end this discussion by observing how quickly the number in Corollary 2.4 grows as we increase the puzzle size. Suppose a $n \times n$ nonogram (i.e. $m = n$) has been created randomly. We round down $\frac{n}{2}$ and $\frac{n}{4}$ to represent the best-case scenario. For an optimistic comparison, we also give the number for a pure exhaustive search.

n	2^{mn}	$\binom{\frac{n}{2}+1}{\frac{n}{4}}^m$
5	3.355×10^7	243
10	1.267×10^{30}	5.766×10^{11}
15	5.391×10^{67}	1.670×10^{26}
20	2.582×10^{120}	1.962×10^{53}

3. Introducing Compressive Sensing

Here, we discuss how we turned a nonogram into a compressive sensing problem. But first, we take a detour to illustrate an approach we had initially taken and comment on its strengths and weaknesses in solving the puzzle.

We will consider the **stylish lambda** as our canonical example:



Initial attempt. Because a nonogram involves an $m \times n$ grid, perhaps the most natural way to represent its solution is as a matrix $X \in \{0, 1\}^{m \times n}$, or by vectorizing it, as a vector $\vec{x} \in \{0, 1\}^N$ with $N = mn$. The (i, j)-th cell of the nonogram would uniquely correspond to x_k, some entry of $\vec{x}$. (We let $k = (i-1)n + j$, for example.) We let $x_k = 1$ to mean that the cell is shaded, and $x_k = 0$ that the cell is empty. The solution to the **stylish lambda** is given by a vector $\vec{x} \in \{0, 1\}^{12}$, where

$$\vec{x} = \left[\begin{array}{ccc|ccc|ccc|ccc} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{array}\right]^T.$$

If we assume that a nonogram had been created randomly, then Lemma 2.1 tells us to expect to shade half of the cells. In other words, the sparsity level s (or the number of nonzero entries) of the solution $\vec{x}$ is equal to

$$s = \frac{mn}{2}. \quad [1]$$

Compressive sensing tells us to provide at least $2s = mn$ measurements for sparse recovery of $\vec{x}$, with an additional factor of $\ln\left(\frac{N}{s}\right)$ for stability. Rather fortunately, this factor becomes a constant $\ln(2)$ and is independent of the puzzle size.

The problem is, it is not obvious how to group cells to faithfully represent the given row and column sequences. Consider the **stylish lambda**. The first row sequence (2) means we must shade a block of 2 consecutive cells. How do we translate this notion of "consecutiveness" when each entry of $\vec{x}$ stands for a single cell? One way is to enumerate all possibilities, e.g. require that

$$x_1 x_2 = 1 \quad \text{or} \quad x_2 x_3 = 1,$$

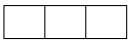
but this creates additional challenges. The degree of such constraint increases with the block size, and the solver we use may not handle "or," a logical operator.

Furthermore, for sequences like (1, 1) and (1, 2) with more than one number, we must be able to also specify that there must exist some number of empty cells between any two consecutive blocks of shaded cells, although how many empty cells, we do not know upfront. Again, we could consider all possibilities, but Theorem 2.3 tells us that that is very unwise.

[1] The exact sparsity level is given by $s = \sum_i r_i$, the sum of all the row sums (or of all the column sums). For analysis, it is more convenient to consider its expectation.

Probably the best we can do is to impose the row and column sequences *weakly*, i.e. we require that the row sums and the column sums are satisfied. These, we can easily do with linear equality constraints; however, there are only $(m+n)$ of them, nowhere close to the $2s = mn$ that is needed for sparse recovery. It seems that we have lost a great deal of information.

Our solution. The primary problem in representing the solution as a vector $\vec{x}$ as described above is that it is hard to group cells and to indicate that, in certain places, there needs to be empty cells. We will come to a new insight by considering the **stylish lambda** and one of its rows, which has $n = 3$ cells:

typical row : 

There are $6 = \frac{3\times 4}{2}$ possible ways to shade a block of cells in this row; mainly,

e_{11} : 

e_{12} : 

e_{13} : 

e_{21} : 

e_{22} : 

e_{31} : 

As shown above, we label these six configurations by

$$e_{kj},$$

where k is the size of the block of shaded cells, and j is the starting column position of the block. Despite the abuse of terminologies, we call these six configurations the **basis vectors** for a row, because every row sequence can be *uniquely* written as a **linear combination** of these basis vectors. For the **stylish lambda**,

	1	1	
	2	1	3
2	■	■	
1			■
3	■	■	■
1 1	■		■

we can represent the first row and the fourth row as

 $= 0\cdot e_{11} + 0\cdot e_{12} + 0\cdot e_{13} + 1\cdot e_{21} + 0\cdot e_{22} + 0\cdot e_{31},$

 $= 1\cdot e_{11} + 0\cdot e_{12} + 1\cdot e_{13} + 0\cdot e_{21} + 0\cdot e_{22} + 0\cdot e_{31}.$

Note that we did not write $1\cdot e_{11} + 1\cdot e_{12} + 0\cdot e_{13} + 0\cdot e_{21} + 0\cdot e_{22} + 0\cdot e_{31}$ to represent the first row, because this violates the rule that two consecutive blocks are always separated by an empty cell. This shows why we get uniqueness.

For each row, we can create these basis vectors, so we insert an index i to denote the row position of a basis vector. Hence, every basis vector has the name

$$e_{ikj},$$

where $i = 1, \cdots, m$, $k = 1, \cdots, n$, and $j = 1, \cdots, n - k + 1$. But often, we will want to focus our attention to one row. In that case, we will suppress the index i for simplicity. Also, we will use e_{ikj} to mean the name of a basis vector as well as the binary variable that indicates whether to shade a block of k cells starting at the (i, j)-th position. It will be obvious from the context to which we refer.

Let us store e_{ikj} as entries of $\vec{x} \in \{0, 1\}^N$, where

$$N = \frac{mn(n+1)}{2}.$$

Hence, the solution to the **stylish lambda** is given by $\vec{x} \in \{0, 1\}^{24}$, where

$$\begin{aligned} \vec{x} &= \left[\begin{array}{cccccccccccccc} e_{111} & e_{112} & e_{113} & e_{121} & e_{122} & e_{131} & \cdots & e_{411} & e_{412} & e_{413} & e_{421} & e_{422} & e_{431} \end{array} \right]^T \\ &= \left[\begin{array}{cccccc|cccccc|cccccc|cccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array} \right]^T. \end{aligned}$$

We call the manner in which we have listed the basis vectors e_{ikj} in the solution $\vec{x}$ the **natural ordering** of the basis vectors.

We see that this approach allows the solution vector $\vec{x} \in \{0, 1\}^N$ to be sparse, since for any row sequence, we will likely need only a few basis vectors to represent the sequence. For the **stylish lambda**, we used 5 basis vectors out of the possible 24. Had a nonogram been created randomly, Lemma 2.1 tells us, on average, each row will have $\frac{n+1}{4}$ blocks of shaded cells, meaning $\frac{n+1}{4}$ basis vectors will be needed. Hence, the sparsity level s of $\vec{x}$ is, on average,

$$s = \frac{m(n+1)}{4}.$$ [2]

This is nearly half of that from our initial approach, meaning we could make fewer measurements and still reconstruct the solution. However, the logarithmic factor for stability is now $\ln(2n)$, which increases (slowly) as the column size increases. But there is a prize in return: we can easily indicate a block of shaded cells now, thanks to the very construction of the basis vectors. Each number in a row sequence identifies a block of shaded cells, which corresponds to a particular basis vector.

By now, we should have noticed that the basis vectors lie along just one direction. In the next section, we will detail how to build constraints from a given nonogram, including those that say empty cells must exist between any two consecutive blocks. We will see that we can easily extract information from the row sequences but not from the column sequences. Therefore, we will impose the column sequences *weakly*, i.e. we only require that the column sums are satisfied.

Of course, we can also create basis vectors for a column so that we make a better use of the column sequences. However, it may be possible that the solution we get using the row basis vectors does not agree with that using the column basis vectors. We do not study here under what condition the two solutions agree.

[2] The exact sparsity level equals $s = \sum_i p_i$, the total number of blocks of shaded cells (row-wise), or equivalently, how many numbers appear in the given row sequences.

4. Designing Constraints

Now that we have laid down the foundations and know how to represent the solution of a nonogram as a sparse vector $\vec{x}$, let us come up with constraints so that we can recover $\vec{x}$ from an l_1-minimization. The only information a nonogram provides are the row and column sequences, so we seek to make the most of them.

As previously mentioned, our basis vectors lie along the row direction, so we will impose the column sequences weakly and require the column sums to be satisfied. This will give us one set of linear equality constraints.

As for the row sequences, we can do much more. We will have one set of linear equality constraints that says how many times a block of size k should appear on each row. Clearly, this is stronger than requiring that the row sums are satisfied. In addition, we will have two sets of linear inequality constraints that enforce the rules of nonogram.

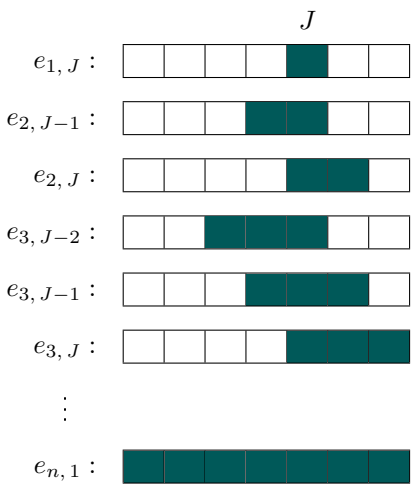
Hence, the l_1-minimization problem we will solve takes the form

$$\begin{aligned} \min_{\vec{x} \in \{0,1\}^N} \quad & ||\vec{x}||_1 \\ \text{subject to} \quad & A\vec{x} = \vec{b} \\ & B\vec{x} \leq \vec{c} \end{aligned}$$

for some matrices $A \in \mathbb{R}^{M_1 \times N}$, $B \in \mathbb{R}^{M_2 \times N}$ and some vectors $\vec{b} \in \mathbb{R}^{M_1}$, $\vec{c} \in \mathbb{R}^{M_2}$. We will determine how large M_1 and M_2 are along the way.

Column sequences. For each column, we want the column sum to be satisfied. So let us focus on one column—the J-th column (where $J \in \{1, \cdots, n\}$). It has the column sequence $(b_1, \cdots, b_p)$ and the column sum $r = b_1 + \cdots + b_p$.[3]

We want to say that there are r many (row) basis vectors that pass through the J-th column. Which basis vectors do pass through the J-th column? For any row, we can (systematically) list the possible basis vectors:



$\Rightarrow$

Consider all e_{kj}'s, where

$k = 1, \cdots, n$

$j = j_{lo}, \cdots, j_{hi}$.

Note,

$j_{lo} = \max\{J - k + 1,\ 1\}$

$j_{hi} = \min\{n - k + 1,\ J\}$.

[3] Before, we associated the letters b and r with a row sequence. But really, there is no distinction between row and column sequences. They are each a sequence of numbers.

So the J-th column sum is satisfied if we require that

$$\sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=j_{lo}}^{j_{hi}} e_{ikj} \;=\; r,$$

where j_{lo} and j_{hi} are defined as in the previous page. Using the natural ordering of the basis vectors, we arrive at a linear system $A_1\vec{x} = \vec{b}_1$, where $A_1 \in \{0,\, 1\}^{n \times N}$. For the **stylish lambda**, we have

$$A_1 = \left[\begin{array}{cccccc|cccccc|cccccc|cccccc} 1&0&0&1&0&1&1&0&0&1&0&1&1&0&0&1&0&1&1&0&0&1&0&1 \\ 0&1&0&1&1&1&0&1&0&1&1&1&0&1&0&1&1&1&0&1&0&1&1&1 \\ 0&0&1&0&1&1&0&0&1&0&1&1&0&0&1&0&1&1&0&0&1&0&1&1 \end{array}\right],$$

$$\vec{b}_1 = \left[\begin{array}{ccc|ccc|ccc|ccc} 3 & 2 & 3 \end{array}\right]^T.$$

It is worth mentioning that we can easily construct A_1 and $\vec{b}_1$ in code. For A_1, we traverse in i, k, and j subject to their limits to determine which of its entries are equal to 1. For $\vec{b}_1$, we read the column sequences and find the column sums. Interestingly, the matrix A_1 does not really depend on the nonogram, in the sense that any other nonogram with the same dimensions will result in the same A_1.

Row sequences. Let us first develop the set of linear equality constraints that specifies how many times a block of size k appears for each row. Again, focus on one row—the I-th row (where $I \in \{1, \cdots, m\}$). From the row sequence $(b_1, \cdots, b_p)$, we can determine t_k, the number of times the number k appears in the sequence.

We get the following n equations for the I-th row:

$$\sum_{j=1}^{j_{hi}} e_{Ikj} \;=\; t_k, \;\; \forall\; k = 1, \cdots, n,$$

where $j_{hi} = n - k + 1$. Using the natural ordering of the basis vectors, we can write the linear system $A_2\vec{x} = \vec{b}_2$, where $A_2 \in \{0,\, 1\}^{mn \times N}$. For the **stylish lambda**,

$$A_2 = \left[\begin{array}{cccccc|cccccc|cccccc|cccccc} 1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ \hline 0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{array}\right],$$

$$\vec{b}_2 = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 \end{array}\right]^T.$$

We merge A_1 and A_2 to get the linear constraint $A\vec{x} = \vec{b}$ for the l_1-minimization problem, where

$$A = \left[\frac{A_1}{A_2}\right] \in \{0,\, 1\}^{M_1 \times N}, \quad \vec{b} = \left[\frac{\vec{b}_1}{\vec{b}_2}\right] \in \mathbb{R}^{M_1},$$

and where $M_1 = (m+1)n$ is the total number of linear equality constraints. Note that M_1 does exceed the minimal number of measurements $2s = \frac{m(n+1)}{2}$ required for sparse recovery. And this is just the beginning.

Row sequences (part 2). Thanks to how we constructed the basis vectors, we can enforce the rules of nonogram by imposing additional linear constraints for the l_1-minimization problem. We will see that these constraints involve inequalities.

One of the rules of nonogram is that we must leave at least one cell empty between any two consecutive blocks of shaded cells. So necessarily, certain pairs of basis vectors e_{kj} of the same row cannot appear at the same time.

For example, we cannot have $e_{11} = 1$ and $e_{12} = 1$ at the same time. If we want a block of 2 cells starting at column 1, then we would be setting $e_{21} = 1$ instead. Since e_{kj} are binary variables, we simply require that $e_{11} + e_{12} \leq 1$ to force one of the two variables to be zero.

For the **stylish lambda**, which has $n = 3$ columns, we have that

$$\begin{aligned}
&e_{11} + e_{12} \leq 1, \quad e_{11} + e_{21} \leq 1, \quad e_{11} + e_{22} \leq 1, \quad e_{11} + e_{31} \leq 1,\\
&e_{12} + e_{13} \leq 1, \quad e_{12} + e_{21} \leq 1, \quad e_{12} + e_{22} \leq 1, \quad e_{12} + e_{31} \leq 1,\\
&e_{13} + e_{21} \leq 1, \quad e_{13} + e_{22} \leq 1, \quad e_{13} + e_{31} \leq 1,\\
&e_{21} + e_{22} \leq 1, \quad e_{21} + e_{31} \leq 1,\\
&e_{22} + e_{31} \leq 1.
\end{aligned}$$

Notice how we systematically listed them using the natural ordering. This gives us the insight of how we would implement these constraints in code.

For each of the four rows, we get these 14 linear inequalities. We can compactly write all 56 of them as $B_1\vec{x} \leq \vec{c}_1$, where $B_1 \in \{0,\, 1\}^{56 \times 24}$ is a matrix of zeroes and ones, and $\vec{c}_1 \in \{1\}^{56}$ is the vector of all ones.

The problem is, we have too many constraints (relative to the size of the solution vector $\vec{x}$) that it would be dishonest to call our approach compressive sensing now. Indeed, an upper bound for the number of these constraints is given by,

$$m \times \frac{1}{2}\left(\frac{n(n+1)}{2} - 1\right)\left(\frac{n(n+1)}{2}\right) \approx \frac{mn^4}{8}.$$

Soon, we will consider row sequences $(b_1, \cdots, b_p)$, where $p > 1$ and the numbers $b_1, \cdots, b_p$ are *all* distinct, and we will derive some constraints for these sequences. So this is what we will do: for the row sequences where some of the numbers are repeated—say, K is repeated—we will impose the condition $e_{Kj} + e_{Kj'} \leq 1$. Again, we do this *only* for the numbers that are repeated, resulting in far fewer (but still necessary) constraints. We end up with the (reduced) constraint,

$$B_1\vec{x} \leq \vec{c}_1.$$

Row sequences (part 3). So far, we have not specified that, given a row sequence $(b_1, \cdots, b_p)$, we must shade a block of b_1 many cells first, then a block of b_2 many cells, and so on. Hence, the order of the blocks matters, but it matters only when the block sizes are not all the same. Take the sequence (1, 1), for example. We cannot tell this sequence apart from the (1, 1) that results from switching the order of the two blocks. With the sequence (1, 2), however, we can.

So what constraints can we use to enforce this rule? For simplicity, let us first consider a row sequence with two numbers,

$$(b_1, b_2) = (k, k'), \text{ with } k \neq k'.$$

We can write the rule as a mathematical statement in this manner:

$$\text{If } e_{kj} = 1 \text{ and } e_{k'j'} = 1, \text{ then } j' > j + k.$$

How did we arrive at this statement? Well, it is clear that, under the hypothesis, we must have $j' > j$, i.e. the second block must appear after the first block. But in fact, we can refine the lower bound with $j' > j + k$, i.e. the second block must appear after the end of the first block. We have a strict inequality because there exists at least one empty cell between the two blocks.

What is really useful for us is the contrapositive statement:

$$\text{If } j' \leq j + k, \text{ then } e_{kj} + e_{k'j'} \leq 1.$$

Hence, given two distinct block sizes k and k', we get several linear inequalities (with $j = 1, \cdots, n - k + 1$ and $j' = 1, \cdots, j + k$) that specify how the two blocks cannot appear. For implementation, we let $j' = 1, \cdots, \min\{j + k, n - k' + 1\}$.

To illustrate this, let us consider a nonogram with column size $n = 4$ and a row sequence (1, 2). What are the linear inequalities we get? Well, if $j' \leq j + 1$, then we have $e_{1j} + e_{2j'} \leq 1$. So just iterate through the possible values of j:

$$\begin{aligned}
j = 1 \quad &\Rightarrow \quad e_{11} + e_{21} \leq 1, \quad e_{11} + e_{22} \leq 1, \\
j = 2 \quad &\Rightarrow \quad e_{12} + e_{21} \leq 1, \quad e_{12} + e_{22} \leq 1, \quad e_{12} + e_{23} \leq 1, \\
j = 3 \quad &\Rightarrow \quad e_{13} + e_{21} \leq 1, \quad e_{13} + e_{22} \leq 1, \quad e_{13} + e_{23} \leq 1, \\
j = 4 \quad &\Rightarrow \quad e_{14} + e_{21} \leq 1, \quad e_{14} + e_{22} \leq 1, \quad e_{14} + e_{23} \leq 1.
\end{aligned}$$

We see that these inequalities do prevent the block of two cells to appear before the block of one cell. Had we solved the l_1-minimization problem with the constraint $A\vec{x} = \vec{b}$ only, we could have ended up with $e_{21} = 1$ and $e_{14} = 1$.

So what do we do when a sequence has more than two numbers? Without loss of generality, consider a sequence of three numbers, $(b_1, b_2, b_3) = (k, k', k'')$. Again, there is no problem if all three numbers are the same; we just have to take care of what happens if at least one number is different from the rest. We want to say:

$$\text{If } e_{kj} = 1, \, e_{k'j'} = 1, \text{ and } e_{k''j''} = 1, \text{ then } j'' > j' + k' \text{ and } j' > j + k.$$

Unfortunately, the contrapositive for this statement cannot be easily expressed as a linear constraint due to the logical operators. Things get even messier when there are two numbers that are the same. Before, we got away with this, because having just two numbers in a sequence implies that they are distinct.

Thus, the only type of sequences we will consider in this part is that with numbers that are *all* distinct. We see that the statement above is true if the following set of statements is true:

$$\text{If } e_{kj} = 1 \text{ and } e_{k'j'} = 1, \text{ then } j' > j + k.$$
$$\text{If } e_{k'j'} = 1 \text{ and } e_{k''j''} = 1, \text{ then } j'' > j' + k'.$$

Since k, k', and k'' are distinct, we can form their contrapositives:

$$\text{If } j' \leq j + k, \text{ then } e_{kj} + e_{k'j'} \leq 1.$$
$$\text{If } j'' \leq j' + k', \text{ then } e_{k'j'} + e_{k''j''} \leq 1.$$

To summarize, we can come up with linear inequalities to ensure that the blocks for a given row appear in the right order, but we can easily do so only when the row sequence involves numbers that are all distinct. Given the column size $n \geq 4$ and a row sequence $(b_1, \cdots, b_p)$ of all distinct numbers, a (crude) upper bound for the number of linear inequalities is given by,

$$\frac{(p-1)(n^2+3n-6)}{2} \approx \frac{pn^2}{2}.$$

We compactly write all the linear inequalities as a single constraint, $B_2\vec{x} \leq \vec{c}_2$, where B_2 is a matrix of zeroes and ones, and $\vec{c}_2$ is the vector of all ones.

Finally, we merge B_1 and B_2, and obtain the linear constraint $B\vec{x} \leq \vec{c}$ for the l_1-minimization problem, where

$$B = \left[\frac{B_1}{B_2}\right] \in \{0, 1\}^{M_2 \times N}, \quad \vec{c} = \left[\frac{\vec{c}_1}{\vec{c}_2}\right] \in \{1\}^{M_2},$$

and where M_2 is the total number of linear inequality constraints. In addition to the dimensions m and n of a nonogram, the number M_2 depends on what kind of row sequences the nonogram has, so we can really only determine how large M_2 is after we finish constructing the linear inequalities.

5. Examples

We present some initial findings below. For each example, the nonogram on the left is the true solution, and that on the right is our recovered solution.

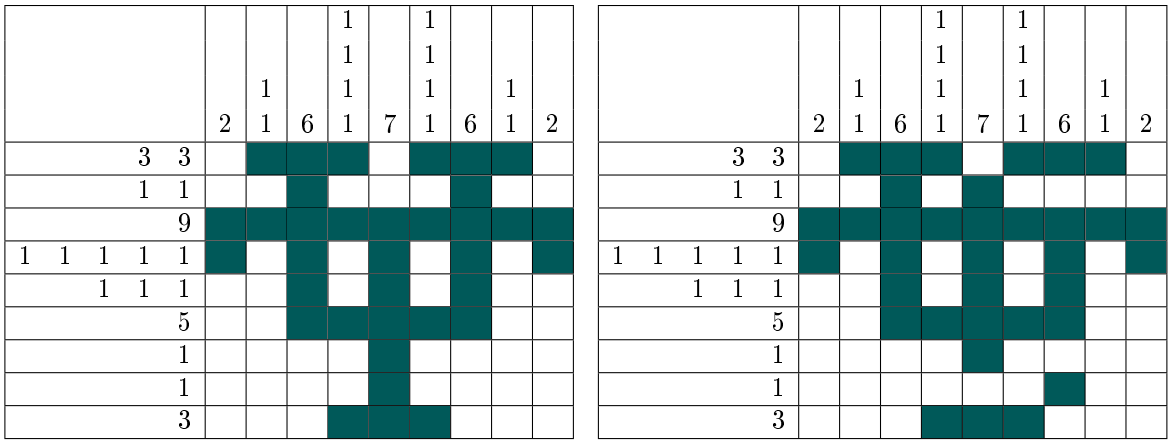
Stylish lambda. 24 unknowns, 15 equality constraints, 2 inequality constraints.



Our solution is flipped, so it may seem like our approach is an incorrect one. But we stress that our solution did in fact satisfy all the row sequences. In particular, the two blocks of one cell did get separated by an empty cell on the fourth row.

The problem, as we can guess, lies with the fact that the column sequences are weakly enforced. We see that the column sum is satisfied for each column, but we do not necessarily have the right blocks occupied.

UT logo. 405 unknowns, 90 equality constraints, 39 inequality constraints.



Here is a larger 9×9 example, where we incur the same mistake we made with the **stylish lambda**. We need to somehow specify that we want a block of 7 cells in the fifth column and a block of 6 cells in the seventh column.

Car. 288 unknowns, 72 equality constraints, 152 inequality constraints.

				4 1	1 1 1	1 1 1	5	1 1 1	1 1 1	2 1 1	3
			5		■	■	■	■	■		
	1	1	1	■			■			■	
		1	5	■			■	■	■	■	■
	1	1	1	■			■				■
	1	3	2	■		■	■	■		■	■
		1	1		■				■		
1	1	1	1	■		■		■		■	
		1	1		■				■		

				4 1	1 1 1	1 1 1	5	1 1 1	1 1 1	2 1 1	3
			5	■	■	■	■	■			
	1	1	1				■		■		■
		1	5	■			■	■	■	■	■
	1	1	1	■			■			■	
	1	3	2	■		■	■	■		■	■
		1	1		■				■		
1	1	1	1	■		■		■		■	
		1	1		■				■		

The row sequence (1, 3, 2) shows that the inequality constraints did a good job at maintaining the order of the blocks with different sizes.

Square root of i. 288 unknowns, 72 equality constraints, 77 inequality constraints.

		1	1	8	1	1	1 1 4	1	1
	6			■	■	■	■	■	■
	1			■					
1	1			■			■		
	1			■					
1	1			■			■		
1	1			■			■		
1	1	1	■		■			■		
	2	1		■	■			■		

		1	1	8	1	1	1 1 4	1	1
	6			■	■	■	■	■	■
	1			■					
1	1			■			■		
	1			■					
1	1			■			■		
1	1			■			■		
1	1	1	■		■			■		
	2	1		■	■			■		

Finally, an example where our method was successful at recovering the solution. We chuckle at the irony that the only one that was successful so far is imaginary. However, we were also lucky, as most columns have the sequence (1).

Note that both the **car** example and this one are of the size 8×8, but we have far fewer inequality constraints here. We see that the number of inequality constraints does depend on what kind of row sequences a nonogram has.

On the contrary, the number of equality constraints depends on only m and n, so we get the same number for both examples.

6. Conclusion

We have come up with a new way to solve nonograms using compressive sensing. Our method is easy to understand and implement, and only requires that we solve a binary integer programming problem. However, it suffers from inaccuracy because we weakly enforce the column sequences.

An easy way to remedy this problem is to do partial fill-ins. For certain columns, we may deduce which cells must be shaded or must be empty. This allows us to "strengthen" the constraints that we get from the column sequences.

Consider the **stylish lambda**. For the third column with column sequence (3), we would normally require the weak constraint,

$$(e_{113}+e_{122}+e_{131})+(e_{213}+e_{222}+e_{231})+(e_{313}+e_{322}+e_{331})+(e_{413}+e_{422}+e_{431}) = 3.$$

But since there are four rows, the column sequence (3) implies that the cells (2, 3) and (3, 3) must be shaded. So we can make a stronger statement by requiring that these constraints are satisfied instead:

$$\begin{cases} e_{213} + e_{222} + e_{231} = 1 \\ e_{313} + e_{322} + e_{331} = 1 \\ (e_{113} + e_{122} + e_{131}) + (e_{413} + e_{422} + e_{431}) = 1. \end{cases}$$

With this simple modification, we *do* get the right solution.

Another possibility comes from the observation that we can also create basis vectors for a column. To put it more simply, we can "transpose" a given nonogram so that the columns become the rows. Then, the existing basis vectors for a row (that is, the existing code) can be used to represent the column sequences instead. This will gives us a second solution. We may want to study under what condition the two solutions agree with each other, and if this implies we have the right solution. If there is a mismatch between the two solutions, we can try to combine them so that we end up with the right one.

References

[1] K. J. Batenburg and W. A. Kosters, A Discrete Tomography Approach to Japanese Puzzles, Proc. Belgian-Dutch Conf. Artificial Intelligence (2004), 243-250.

[2] K. J. Batenburg and W. A. Kosters, Solving Nonograms by Combining Relaxations, Pattern Recognition **42** (2009), 1672-1783.

[3] Sancho Salcedo-Sanz, et al., Solving Japanese Puzzles with Heuristics, Computational Intelligence and Games (2007), 224-231.

[4] Jinn-Tsong Tsai, Solving Japanese Nonograms by Taguchi-Based Genetic Algorithm, Applied Intelligence **37** (2012), 405-419.

[5] Nobuhisa Ueda and Tadaaki Nagao, NP-Completeness Results for NONOGRAM via Parsimonious Reductions, technical report (1996).

[6] Jan N. Van Rijn, Playing Games: The Complexity of Klondike, Mahjong, Nonograms and Animal Chess, master's thesis (2012).

[7] Gerhard J. Woeginger, The Reconstruction of Polyominoes from Their Orthogonal Projections, Information Processing Letters **77** (2001), 225-229.

[8] Jan Wolter, Survey of Paint-by-Number Puzzle Solvers, http://webpbn.com/survey/, accessed Nov. 2012.