Formal description of Thesis problem

Raphaël P

18/12/2014

1 Description

Regular Nonograms are puzzles consisting of a set of x horizontal lines (rows) and y vertical lines (columns) (x may equal y) representing a grid. The aim is the to colour certain squares from the grid, according to specific rules, to obtain the final Nonogram, which should represents something recognisable (a drawing, or a letter, ...). In order to know which squares must be coloured and which must be left blank, for each column and row of the grid, a *description* is given. A description is a sequence of integers $a_1, a_2, ..., a_i$, where $a_n <= y$ for a row and $a_n <= x$ for a column. The description details which of the square within the current row or column must be coloured. For an arbitrary description a_0, a_1, a_2, the user knows that three blocks of a_0, a_1 and a_2 consecutive squares must be coloured (in that order), call them block0, block1 and block2. Moreover, there can be zero or more blank squares before block0 (so between the first boundary and the start of the first block), then there must be one or more blank squares between block0 and block1 , same for between block1 and block2, and finally there must be zero or more blank squares from the end of block2 and the side of the grid. We can use regular expression syntax to describe a whole row or column. Let us define that a blank square is labeled 0 and a coloured square is labeled 1. Given a description a_0, a_1, a_2 for a given row or column, we can describe the whole row or column as such:

$$0^{\star}1^{a_0}0^{+}1^{a_1}0^{+}1^{a_2}0^{\star} \tag{1}$$

Given descriptions for all columns and all rows, the user can colour the appropriate squares and solve the puzzle.

A lot has already been figured out about regular Nonograms (described above), therefore, my task is to study non-regular Nonograms, namely Nonograms based on curves as opposed to horizontal and vertical lines. All rules will keep on applying, except the format of the description. Descriptions will be given for a particular curve as opposed to a row or column in the case of regular Nonograms. A curve will have a description for its left side and one for its right side. A description will detail how many and in what order will the faces (not talking about squares anymore) adjacent to the curve must be coloured. Each curve will have two descriptions.

I believe a couple of intermediary steps are needed, in order to delve into Nonogram (Curvograms, not official name) creation step by step:

- tangram Nonograms, 4 slopes only
- Arbitrary number of slopes
- Approximated curves Nonograms (using sequences of straight lines)
- Using Bézier curves (FINAL)

2 Four projects

2.1 Tangram Nonograms

Tangram Nonograms consist of a set of lines with in total four different slopes. A line can either be horizontal, vertical, 45 degrees of -45 degrees. This is the very first subproblem i consider as it is very simple in complexity, resembles features of the regular Nonograms, and yet some features of Curvograms.

2.1.1 What makes a Tangram Nonogram

Let us describe what is a Tangram Nonogram

- An original drawing consisting of lines, each with a horizontal or a vertical or a +45 or a -45 slope.
- A set of four lines representing the bounding box (frame of the puzzle). It cannot be too large, as that would render the original drawing too small and hard to colour. It cannot be too small, as it must contain the whole original drawing plus a buffer length.
- Possibly another set of lines with slopes: horizontal, vertical, +45 or -45 in case the original drawing is too easily identifiable.
- For each line, a left description and a right description.
- Each line must be extended to the bounding box
- Each face will be a convex polygon!/ with max complexity a hexagon (6 sides)

2.1.2 Input file format

Regarding the original drawing, we need to make sure that the input format helps as much as possible. For regular hole-less pictures, the input file could be a set of coordinates representing every vertex of the drawing in a **counter-clockwise** order (no issues with points that are part of the drawing multiple times). From such a file, the picture could easily be reconstructed and the lines extended which would satisfy the first point of what consists a Tangram Nonogram. We use a set order to specify the vertices of the drawing to avoid encountering problem based on the shape of the drawing (DEGENERACIES)

Problems with this method arise when holes are taken into account. As a matter of fact, using the previous method, it becomes complex to figure out which polygons are holes and which are not (we would have to compare all coordinates of all vertices in a hole and check whether that polygon actually lies within another polygon, if so then it is a hole. In the Nonogram application, it would not be useful to have polygon within polygon which are all part of the drawing. Yet comparing all the vertices is a lengthy and cumbersome tasks.

Here is a better idea:

The input file can be a set of coordinates representing multiple loops, where a loop is a polygon or set of polygon with the same depth. The depth is whether or not the polygon is part of the original picture. We consider the depth because of holes. A polygon with a hole, itself containing a polygon can be thought of a sequence of three loop. The main loop, (loop 1) consists of the coordinates of the main polygon, it has depth 1. The second loop is the hole present within the main polygon, it has depth 2 and loop 2 is the set of coordinates of the hole. The last polygon, within the hole but part of the original picture has depth 3 and its loop is the set of coordinates defining it. With this definition, all odd numbered depth polygons are part of the original drawing and even numbered depth polygons are holes (not part of the original drawing). FIND A WAY TO REPRESENT THOSE. (maybe a set of coordinates with some indication for the depth of the current polygon)

Input file can also be an 'svg' file. Those are XML/HTML files with information about the shapes. Here is a svg file for a polygon with 5 sides:

```
<!DOCTYPE html>
<html>
<body>

<svg height="400" width="500">
  <polygon points="220,10 300,210 250,400 170,250 123,234"
 style="fill:lime;stroke:purple;stroke-width:1" />
  Sorry, your browser does not support inline SVG.
</svg>
```

```
</body>
</html>
```

We can work on python with svg files, using the svgwrite module. Already installed using pip. Here are the basic functionalities:

resources:https://pythonhosted.org/svgwrite/

```
import svgwrite
dwg = svgwrite.Drawing(filename='test.svg',
size=("800px","600px"))
# main function to add to the current svg fill defined above
dwg.add(dwg.polygon(points=[(x,y),(x,y),...],
stroke_width="1",
stroke="black" # can also use rbg
fill_opacity = "0.0" # how opaque (0.0 to 1.0)
fill"blue" # filling colour))
dwg.save()
```

Using these could be beneficial because you can already tell the shape represented from the header, 'polygon'

Several conditions must be met:

- Faces 'area' should be similar (up to a constant factor) We do not want very small or very large faces
- Intersection points should be far away enough from one another (section 2.6)
- do we allow for 3/4 line intersections? With only 4 different slopes, we can obtain at most a four line intersection. The problem then becomes to know which face is adjacent to the intersection: the edge adjacent faces (only 2) or the vertex adjacent faces (all 4). ASK PEOPLE FOR WHAT THEY THINK.
- We dont want curves/lines that do not do 'anything', namely each curve should cross the pictogram at least once
- We might want to curves/lines to be somewhat uniformly distributed in the bounding box
- the Nonogram should be uniquely solvable (only one image can results from solving it)
- the Nonogram should reflect the input difficulty (2.7)
- From the bare Nonogram, the final image should not be visible.
-

2.2 Arbitrary slopes

2.3 Approximated Curves

2.4 Bézier Curves

2.4.1 Local Operations

These are the operations used to either extend a curve to the bounding box or glue it to some other curve, in order to make the Nonogram more complex/ interesting.

The curves to be used are Bézier curves. A section of the curve is defined by 4 points, two being the end points of the section and the other two being the direction in which the curves go.

I shall start with using polygonal lines instead of Bézier curves, as they are easier to work with. As a matter of fact, looking at straight lines, with four different slopes: vertical line, horizontal line, diagonal + line, diagonal - line. Diagonal lines have slopes of $45°$

2.5 Fatness

The Fatness of a facet can be described in several way. One of them is to look at the closest enclosed circle (radius r) and the smallest bounding circle (radius R) and compute the fatness as the ration $\frac{r}{R}$

2.6 Intersection distance

how far away should intersection points be. Variable using the programme to see what kind of Nonogram are obtained depending on its value.

2.7 Difficulty of Nonogram