penpal: Finding the Two Identical Penmen

Waseem S. Daher
wdaher@mit.edu

February 16, 2006

Abstract

"Find The Two Identical Penmen" is a poster, created by Gary Blehm, with 1,047 stick-figure-like drawings known as penmen. The goal of this project was to create software that would find the identical penmen, or at least substantially simplify the task of finding the two penmen, to the point where it would be quick and easy for a human to identify the correct match. In this paper, I discuss the design and implementation, in the Python programming language, of a straightforward and reasonably simple algorithm that is fairly reliable in accomplishing this goal: it correctly identifies the identical penmen in the poster, in its first few guesses.

1 Introduction

One of my friends owns the 1,047-penman "Find the Two Identical Penmen" poster, and put it up on her door. One night, in early September, 2005, a bunch of us were in a lounge in my dormitory, and the poster became the topic of discussion.

People had already seen the poster, and some had even tried to find the identical penmen by hand, though most of them gave up fairly quickly. Various suggestions were offered, along the lines of "Well, you can rule out all the penmen doing strange things[1], since it would be too easy to find two of those, because they catch your eye."

While this is certainly a valid point, I was still unconvinced that you could find the penmen quickly, even given this strategy. So, as an undergraduate studying computer science, I proposed a different solution: "Guys, computers can do anything these days — I could easily write a program to find the identical penmen."

The claim was met with skepticism, though. Not skepticism at the technology, but skepticism that I would actually follow through with such a grandiose statement. Feeling invigorated by the challenge, I made a vow: "The identical penmen will be found before this year's senior class graduates!" (that is to say, May 2006).

Unfortunately, the start of the Fall semester, and the courses that come with it, hampered my progress. But in about two days in January, I took a photograph of the poster, wrote some software, and found the penmen. This is my story.

[1]For example, the one having water dripped on its head with a dropper, in Figure 1.

Figure 1: An arrangement of penmen, created for testing.

2 The Algorithm

Since I didn't actually start the actual project until January, I had months to think about an algorithm. After much discussion with just about anyone who was willing to listen, I actually implemented my first idea... and it didn't work. With a bit more refinement, though, the second attempt succeeded, and is detailed below.

2.1 Overview and Performance

The algorithm basically has two parts: First, segment out and identify each individual penman. Then, compare each against every other one, and assign each pair a score. Finally, pick the pair with the best score, and offer that up as the best match.

From an asymptotic standpoint, this requires $O(n)$ time for segmentation, and then approximately $n(n-1)/2$ operations for comparison, for an overall runtime of $O(n^2)$, where n is the number of penmen in the poster. It is worth noting that this algorithm only needed to run (and be successful) once, so the asymptotic runtime is not as much a concern as the *actual* runtime.

The actual implementation was run in Python 2.4.2, using the Python Imaging Library 1.1.5, on a 2.8 GHz Pentium IV laptop with 512 MB of RAM, running Ubuntu "Breezy

Badger". The computation finished in approximately five minutes, processing my 1626x1873 black-and-white JPEG image of the poster.

2.2 Preprocessing

I took a photograph of the poster with my digital camera, for use in this program. There are a few problems with this technique, though: the image ends up slightly skewed, and the image is not black-and-white, as desired. To address the second problem, I edited the image using The Gimp, to crop out everything except for the penmen, and then increased the brightness/contrast until it looked good, and then converted it to black-and-white.

2.3 Segmentation

The first real step in the algorithm is to identify each individual penman. The strategy is as follows: Scan the image until you find the first black pixel. Perform a flood fill on this pixel, that is to say, select all adjacent black pixels, and select all black pixels adjacent to those, etc. The object you have selected is one of the following: a whole penman, a penman's head, a penman's body, or a random small floating object (The problem is made slightly more difficult by the fact that penmen heads and bodies are not necessarily connected).

Now, compute the centroid, width, and height of the object you have selected, and store those values in an array. Repeat until you have scanned the entire poster.

Next, we need to discard small objects so that we are left only with full penmen or just penman bodies. Average all the widths and heights, and then go through and remove all objects whose widths or heights are smaller than the average (because we think they are heads or random floating objects that aren't penmen). At this point, the penmen are ready for comparison.

2.4 Comparison

We use a fairly naive and slow $O(n^2)$ strategy of comparing every penman against every other penman. One could imagine something more elaborate like defining a function on penmen that gauged certain attributes, and then the two identical penmen would be the penmen with values that were closest together (a strategy that could work in $O(n \lg n)$ time), but this seemed a bit like overkill.

The next question is: What function do we use for comparison/determining similarity? I chose to draw a bounding box of average penman size (computed by multiplying the average widths and heights by some tweaked constants) around each penman centroid, as in Figure 2, and then XORing those boxes with each other. Then, I counted the number of differences in the result, and assigned that as the score for the pair. One could imagine many other techniques, like performing some sort of 2-D convolution/correlation on the images, which would probably be more resilient to small changes.

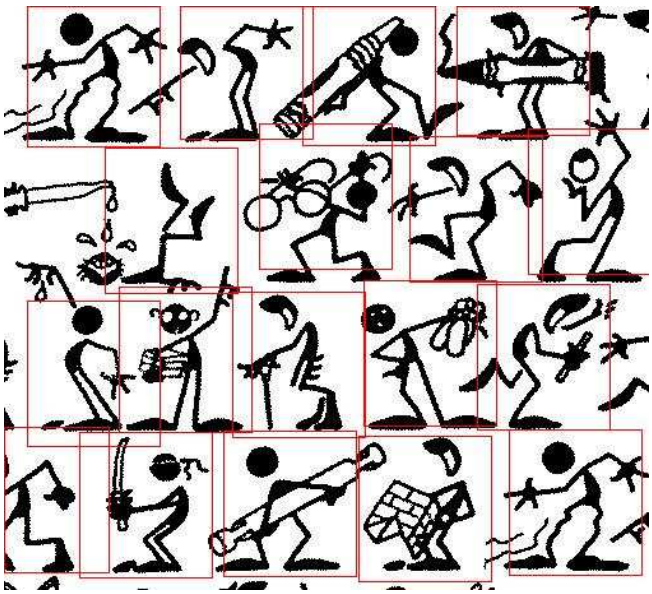

Figure 2: Penmen with bounding boxes drawn in.

2.5 Identifying the match

Since every pair of penmen is assigned a score that would be zero if they were identical, and large if they were completely dissimilar, to find the identical penmen, we need only examine the pairs of penmen in ascending order by score. In practice, the identical penmen are usually in the top 20 pairs (on my trial image, the identical pair was the 15th one). Image noise and near-identical penmen drawn in to fool the viewer typically cause the identical pair not to be the first.

2.6 Assumptions and Effectiveness

"Identical" is, unfortunately, a fairly ambiguous word. In designing the algorithm, a number of assumptions were made. Perhaps most critically, we assume that the identical penmen would not be different sizes or rotated differently. It turns out that this assumption was valid, but it doesn't necessarily follow from the statement of the problem.

There are a few failure modes for the algorithm. Most notably, if two penmen are actually touching, the algorithm thinks of them as one large penman. This could be problematic if one of the identical penmen was touching something, and one wasn't. In practice, the algorithm only works well if the two identical penmen do not have many pixel differences between them.

3 Conclusions

In the end, the program accomplished its desired goal: it made the process of finding the penmen dramatically easier. While the first match it returned was not correct, the correct answer did lie in the top 20 pairs, which is certainly within the realm of numbers of pairs that can be examined by a human.

4 Acknowledgements

This project would not have been a success without the help of many people.

First, I would like to thank Joshua Mandel for providing me with a lot of good advice and helpful tips for the algorithm applied here; I probably would have tried something much more complicated and clumsy were it not for his help.

Next, I would like to thank Kiera Henning, Keith Winstein, Mika Tomczak (actual owner of the poster in question), Ray He, Caitlin Murray, Alice Macdonald, Professors David Karger and Frédo Durand of MIT, and all of the members of MIT French House for listening to my algorithm proposals and offering useful advice. Caitlin offered up a novel approach that probably would have been even better: Email a million people with two pieces of the poster, asking: "Are these two images identical? If so, email back and win a prize!" — the ultimate distributed computing solution.

Of course, this would also never have happened without Gary Blehm, the poster's creator. I sent him an email before I began the project, and again when I finished it — he was very enthusiastic about it and proved quite helpful.

5 Appendix

5.1 How to run the code

Two steps are required to process the penmen, and one step to view your results:

```
$ python segment.py yourimage.jpg
$ python compare.py penmen-data.txt
# Now, look at the output of penmen-comparison.txt
# and view the pairs with the following command
$ python show.py penmen-data.txt
```

5.2 `segment.py` Source Code

```
#!/usr/bin/python

# segment.py
# Part of the "penpal" project
# (Finds Gary Blehm's Identical Penmen)
# http://web.mit.edu/wdaher/www/penpal/
#
# Waseem Daher
# Jan. 21, 2006

import Image, ImageChops
import os, sys
import time

#infile = '4x4-identical-bw.jpg'
infile = sys.argv[1]
datafile = 'penmen-data.txt'

# What value do we consider our
# threshold for 'black'?
BLACK_THRESHOLD = 128
# Throw away objects whose dimensions
# are TOO_SMALL times the average
TOO_SMALL = 1
# Throw away objects w/o enough pixels
TOO_FEW_PIXELS = 1
# When selecting a bounding box, make it
# BOUNDING_BOX times bigger than the average
# penman
BOUNDING_BOX_W = 2.5
BOUNDING_BOX_H = 3
# What percentage of the avg. to move
# up the y-val of the centroid?
# (because we want to capture the head)
HEIGHT_SHIFT = 0.3
# Magic numbers are a hack, but hey
BIG_NUMBER = 99999999999

class Penman:
    def __init__(self):
        self.points = []
        self.centroid = None
        self.dims = None
    def addPoint(self, point):
        self.points.append(point)
    def getCentroid(self):
        if self.centroid is None:
            self._getCentroidAndDims()
        return self.centroid
    def getDimensions(self):
        if self.dims is None:
            self._getCentroidAndDims()
        return self.dims
    def getNumPixels(self):
        return len(self.points)
    def _getCentroidAndDims(self):
        sumx = sumy = 0
        # Magic numbers are sort of a hack,
        # but perhaps this is faster than
        # doing an or comparison every time
        minx = miny =  BIG_NUMBER
        maxx = maxy = -BIG_NUMBER
        for point in self.points:
            (x,y) = point
            if x < minx:
                minx = x
            if x > maxx:
                maxx = x
            if y < miny:
                miny = y
            if y > maxy:
                maxy = y
            sumx += x
            sumy += y
        avgx = sumx / len(self.points)
        avgy = sumy / len(self.points)
        self.centroid = (avgx,avgy)
        self.dims = (maxx-minx, maxy-miny)

def getNeighbors(x,y,rows,cols):
    """
    Returns the set of points
    that are the neighbors of the
    supplied point
    """
```

```
    neighbors = []
    if x > 0 and y > 0:
        neighbors.append( (x-1,y-1) )
    if x > 0:
        neighbors.append( (x-1,y) )
    if x > 0 and y<rows-1:
        neighbors.append( (x-1,y+1) )

    if y > 0:
        neighbors.append( (x,y-1) )
    if y < rows-1:
        neighbors.append( (x,y+1) )

    if x < cols-1 and y > 0:
        neighbors.append( (x+1,y-1) )
    if x < cols-1 and y > 0:
        neighbors.append( (x+1,y) )
    if x < cols-1 and y<rows-1:
        neighbors.append( (x+1,y+1) )

    return neighbors

def findPenman(x,y,im):
    """
    Given a starting black pixel, find the
    maximally-sized connected region of
    pixels. This is a destructive read -
    the pixels get set to 255 after we
    go over them.
    """
    (cols,rows) = im.size
    me = Penman()
    stack = [(x,y)]
    while len(stack) > 0:
        (a,b) = stack.pop()
        # Blank the pixel so we don't try to
        # process this one again
        im.putpixel((a,b), 255)
        me.addPoint((a,b))
        # Check all the neighbors
        # and add them if they are
        # black.
        neighbors = getNeighbors(a,b,rows,cols)
        for (i,j) in neighbors:
            try:
                if im.getpixel((i,j))<BLACK_THRESHOLD:
                    stack.append((i,j))
            except:
                print "Size is:", im.size
                print "Tried to get:", i, j
    return me

im = Image.open(infile)
(cols, rows) = im.size

average_width = 0
average_height = 0
average_num_pixels = 0
myPenmen = []
for x in range(cols):
    for y in range(rows):
        if im.getpixel( (x,y) ) < BLACK_THRESHOLD:
            man = findPenman(x,y,im)
            w,h = man.getDimensions()
            average_width += w
            average_height += h
            average_num_pixels+=man.getNumPixels()
            myPenmen.append(man)

#--- Throw away "penmen" that don't make
#    the cut (e.g. too small)
average_width /= len(myPenmen)
average_height /= len(myPenmen)
average_num_pixels /= len(myPenmen)

min_num_pixels = average_num_pixels * TOO_FEW_PIXELS
min_width = average_width * TOO_SMALL
min_height = average_height * TOO_SMALL
bounding_box_width = average_width * BOUNDING_BOX_W
bounding_box_height = average_height * BOUNDING_BOX_H

realPenmen = []
for man in myPenmen:
    w,h = man.getDimensions()
    numPix = man.getNumPixels()
    if w > min_width and h > min_height \
        and numPix > min_num_pixels:
        realPenmen.append(man)

# Great, we have all the real penmen
# Dump this to a list.
file = open(datafile, 'w')
lines = []

lines.append('# Penman image details')
lines.append('# Date: %s' % time.asctime())
lines.append('infile = "%s"' % infile)
lines.append('min_width = %i' % min_width)
lines.append('min_height = %i' % min_height)
lines.append('average_width = %i' % average_width)
lines.append('average_height = %i' % average_height)
lines.append('bounding_box_width = %i' \
            % bounding_box_width)
lines.append('bounding_box_height = %i' \
            % bounding_box_height)
lines.append('num_penmen = %i' % len(realPenmen))
lines.append('')
lines.append('# Penman data')
lines.append('# Number, centroid_x, centroid_y')
for i in range(len(realPenmen)):
    centroid_x,centroid_y = realPenmen[i].getCentroid()
    lines.append('%i,%i,%i' % (i, centroid_x, \
        centroid_y - (HEIGHT_SHIFT * average_height)))

file.writelines([foo + '\n' for foo in lines])
file.close()

#os.system('cat %s' % datafile)
```

5.3 compare.py Source Code

```
#!/usr/bin/python

# compare.py
```

```
# Part of the "penpal" project
# (Finds Gary Blehm's Identical Penmen)
# http://web.mit.edu/wdaher/www/penpal/
#
# Waseem Daher
# Jan. 21, 2006

import Image, ImageDraw, ImageStat, ImageChops
import os, sys
import time

outfile = 'penmen-comparison.txt'

filename = sys.argv[1]
data = open(filename, 'r')
lines = data.readlines()
vars = lines[:11]
# Import our variables again
# Yeah, this is totally not secure,
# but whatever, this isn't a secure
# application
for line in vars:
    exec(line)

# Now, read in the penmen data
penmen = [None] * num_penmen
info = lines[13:]
for line in info:
    num,x,y = eval(line)
    penmen[num] = (x,y)

im = Image.open(infile)

def getMan(image, centroid, width, height):
    imgx,imgy = im.size
    centroidx,centroidy = centroid

    tlx = max(0, centroidx - width/2)
    tly = max(0, centroidy - height/2)
    brx = min(imgx, centroidx + width/2)
    bry = min(imgy, centroidy + height/2)

    return image.crop( (tlx,tly,brx,bry) )

def compare(man1, man2):
    diff = ImageChops.difference(man1, man2)
    stat = ImageStat.Stat(diff)
    return stat.sum[0]

print "Loading all penmen..."

for i in range(num_penmen):
    penmen[i] = getMan(im, penmen[i],
                       bounding_box_width,
                       bounding_box_height)

print "All penmen loaded, beginning comparison..."

scores = {}
for i in range(num_penmen):
    for j in range(i+1, num_penmen, 1):
        myscore = compare(penmen[i], penmen[j])
        if scores.has_key(myscore):
            scores[myscore].append( (i,j) )
        else:
            scores[myscore] = [(i,j)]

# Now sort and print out in order of goodness
oldStdOut = sys.stdout
file = open(outfile, 'w')
sys.stdout = file

print "Penmen run"
print time.asctime()

actual_scores = scores.keys()
actual_scores.sort()
for act in actual_scores:
    print act, scores[act], "\n"

# Restore old stdout
sys.stdout = oldStdOut
file.close()
```

5.4 show.py Source Code

```
#!/usr/bin/python

# show.py
# Part of the "penpal" project
# (Finds Gary Blehm's Identical Penmen)
# http://web.mit.edu/wdaher/www/penpal/
#
# Waseem Daher
# Jan. 21, 2006

import Image, ImageDraw
import os, sys
import time

outfile = 'show-image.jpg'

filename = sys.argv[1]
data = open(filename, 'r')
lines = data.readlines()
vars = lines[:11]
# Import our variables again
# Yeah, this is totally not secure,
# but that's fine here
for line in vars:
    exec(line)

# Now, read in the penmen data
penmen = [None] * num_penmen
info = lines[13:]
for line in info:
    num,x,y = eval(line)
    penmen[num] = (x,y)

os.system('head -2 %s' % filename)

im = Image.open(infile)
im = im.convert('RGB')
```

```
def goBound(image, centroid, width, height):
    draw = ImageDraw.Draw(im)
    imgx,imgy = im.size
    centroidx,centroidy = centroid

    tlx = max(0, centroidx - width/2)
    tly = max(0, centroidy - height/2)
    brx = min(imgx, centroidx + width/2)
    bry = min(imgy, centroidy + height/2)

    draw.rectangle([(tlx,tly),(brx,bry)],
                   outline="red")


freshImage = im.copy()

while True:
    number = input("Draw on (-1 to reset," \
                 + " -2 to draw on all," \
                 + " -3 to quit," \
                 + " -4 to draw): ")
    if number == -3:
        break
    elif number == -1:
        im = freshImage
    elif number == -4:
        im.save(outfile)
        os.system('display %s &' % outfile)
    elif number == -2:
        for i in penmen:
            # Draw the bounding box
            goBound(im, i, bounding_box_width,
                    bounding_box_height)
    else:
        goBound(im, penmen[number],
                bounding_box_width,
                bounding_box_height)
```