TJHSST Computer Systems Lab Senior Research Project
Reverse Engineering Graphs: Obtaining Data Points from Scatterplots
2008-2009

Maya Wei

June 10, 2009

Abstract

Various programs exist to take data points and use them to render a graph. However, once the data are put into visual form, there is a loss of numerical information if the original data cannot be obtained. This project seeks to take data from a graph; in essence, the purpose is to reverse engineer a given graph. This will provide for a set of data points which can be used for various other numerical purposes, not simply the graph form in which they are presented. The purpose of this project was to investigate basic forms of image analysis while being able to display the results in a streamlined fashion. This program was chiefly an exploration into the ideas of visual analysis without the use of any specific theories.

Keywords: image analysis, graphs

1 Introduction

1.1 Scope of Study

A scatterplot is a visual representation of bivariate data. It is used so that the human eye may more easily identify trends in data. However, once the data are in this visual form, there is a loss of direct numerical information. If provided only a graph, there is no way to perform statistical tests upon dots laid along an arbitrary set of perpendicular bars. By reverse-engineering a graph, the user is able to derive the base information stored within that graph. A computer cannot as easily see and identify axes, points, and numbers as we do, but also has the capability to derive points faster and with more precision than a human.

In trying to read a graph, various image

analysis techniques will be used: edge analysis and number identification are the two most prominent factors. The intention of this project is to develop methods which will be capable of successfully reading a point's location to its axes. The program has not accommodated the various graph-creating programs' (OpenOffice Calculate, Microsoft Excel) use of different formatting with different colored backgrounds, different guidelines, and different shapes of points; rather, we are assuming that the graphs read are minimalist in design and are not "chart ink" squanderers.

2 Background Information

Image analysis is the focus of a generally 2-D surface, dealing with issues such as connectivity and perceived depth. The field of image analysis and computer vision is highly advanced at this point in time. There have been many papers written on shape identification, image recognition, and graphic rendering; there are various methods developed for edge-detection. While there is no project similar to what is being done here with graphs and data points, undoubtedly somebody has sought to do it before; with effort, no doubt it was a success. Initial research showed that a patent had been developed for an item that would manually scan a handwritten graph and convert it to data from there.

Much progress has been made in the field of document image analysis; the result is what we can see on programs such as Google Books. By first being able to separate specific letters through pixel analysis – finding "connected pixels" in order to read letters and ultimately words. Using image analysis, it is thereby possible to digitally recontruct a hard copy of a book; the copy would be searchable, and would recognize images within the hard copy as well. My scanner attempts to do this with pdf files; the technology is fairly universal now and applies to items of interest such as handwriting. Dr. Andries van Dam of Brown University, whose focus is on user interfaces, is in the process of developing a math program that can convert handwritten numbers, functions, and other equations drawn on a tablet to numerical equations in Mathematica. The identification process is complex, using the previous letters to try and identify the latter; however, part of his project is rooted in image analysis.

Edge detection techniques are varied. Many of them use brightness as a measure of contrast; if there is a point considerably brighter than another, one can assume that the brighter is an "edge." John F. Canny, in 1986, developed a method of edge detection used to this day; his algorithm involved noise reduction, gradients, relative intensity, and setting certain thresholds such that high differences in intensity are first identified, and then lower differences are identified that could be softer parts of the edges.

The field of image analysis is expansive; the

intent of the project is to scrape at the surface of image analysis and see what methods I could develop on my own.

3 Procedures and Methodology

This project utilizes Java. The program can be divided into three separate parts: edge detection, subgrouping, and image recognition. A fourth part, scale compilation, was not reached in the project, but will be discussed in its theoretical basis.

The input data is to be found in the form of a graph in a png file. The image being used has been generated by OpenOffice Calc, displaying a graph with points at (1, 1), (2, 2), (3, 3), (4, 4), and (5, 5). The graph is minimalist; it has no background color, no guidelines. These attributes detract from the graph as a whole, putting in needless information; it is also easier for the program to deal with the smallest amount of extraneous data possible. See Figure 1 for the graph.

3.1 Edge Detection

The edge detection in this program is basic, given that there is no multi-colored background - rather, everything is white. The program takes the color of the point at (0, 0) and makes the assumption that this point's color is the same as the background color. That is, if (0, 0) is white, the background for the rest of the graph should also be white.

The program then looks at each individual point (x, y) and searches its adjacent neighbors (left, right, up, down, and diagonals). If any of the points' neighbors are the same color as (0, 0), then it can be implied to mean that (x, y) is an edge. If the point is found to be an edge, then it is stored in an ArrayList edgeList. The int[] is a two-value array of [x coordinate, y coordinate] for all necessary points. See figure 2 for a visual illustration.

3.2 Subgrouping

Once the edged-points are all placed within edgeList, there is a task to split this ArrayList into something more applicable. These edge points belong to three separate categories: some points belong to a number on the scale; some edge points belong to the axis; some edge points belong to the graph-points. The edged-points must be divided accordingly, as their functionality in future methods depends on their being properly sorted. However, they must be sorted by how they are connected first.

Each point is placed into another, smaller, ArrayList. These ArrayLists hold only points that are connected to each other by immediate neighbors – (x+1, y), (x-1, y), (x, y+1), (x, y-1). Iterating through the entire edgeList, each point is placed with its neighbors in an arduous search. If there are no adjacent pixels that are edges that have already been placed in an ArrayList, a new ArrayList is added. Initially, an iterator

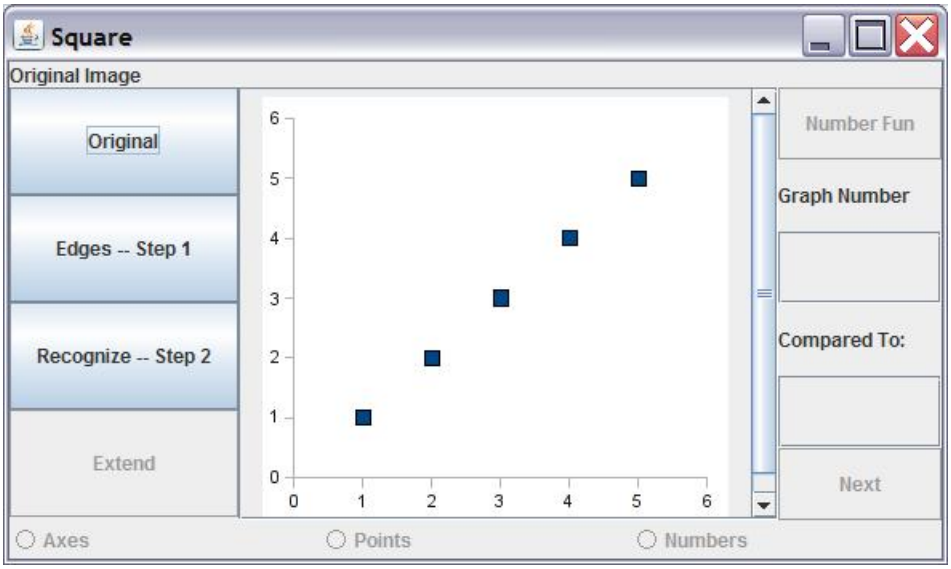

Figure 1: Figure 1. The standard graph used through the project; created in OpenOffice Calc.

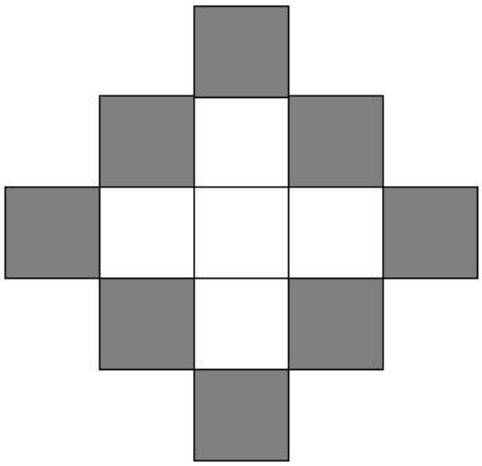

Figure 2: Figure 2. The points in dark grey are considered the edges; they touch the outside (the white). The inner points do not touch the outside white, even if they are a different color, and thus are not considered an edge.

was used; however, iterators do not allow for the modification of the ArrayList while the iterator traverses through. Thus, finding all connected points uses an iterative method (do-while loop) to search through edgeList for each point's neighbors, removing that point from edgeList once it's found.

Once again, it is necessary to sort the grouped-arrayLists into one of three categories: Points, Numbers, or Axis. The axis, in this situation, is recognized as the longest collection of the grouped-arrayLists, as grouped-points and grouped-numbers don't have that much of a collective perimeter. Using this assumption of the axis, the grouped-points located to the right of the x axis are Points (on the graph), whereas the connected points to the left of x and below the y are numbers.

As seen in Figure 1, the axes have tick marks. These tick marks do not detract from the graph, nor to the gathering of information, but it was also possible to filter out the tick-marks when highlighting the axis by using the longest straight lines; the x and y axis were also defined by using the longest lines within the axis point.

When the Points are determined, it is possible to identify the centers of such points. All it requires is the uppermost, bottommost, leftmost, and rightmost coordinates in each Point ArrayList; then the average can be taken. From there, it is possible to extend lines in order to show the relationship of the Points to the Axis, as shown in Figure 3.

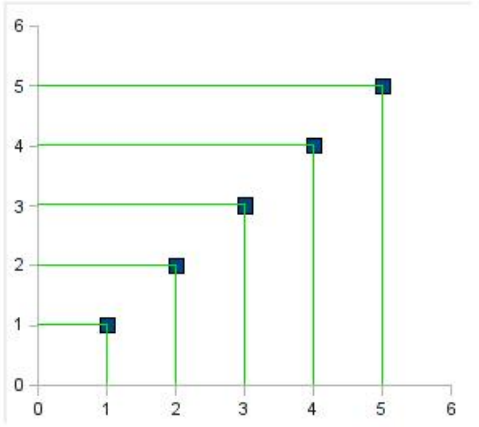

Figure 3: Figure 3. Extended Lines.

3.3 Image Recognition

The second portion of the program was made in intent to analyze the specific number values on the scale of the axes. Seeing as the computer cannot immediately recognize a system of pixels to represent a quantitative figure, it is necessary to make a program that could simulate such.

Each Number is parsed from the image; its subimage is taken out and subsequently modified. Each subimage, when parsed, is expanded to a 50x50 image. Every image is closely parsed, going only as far out as the furthest pixel goes. Then, in order to increase the number of pixels (and because when images expand, there is a blur about them), the image is changed entirely to black-and-white. If there is a pixel that is not exactly white, it becomes black. This can be seen in Figure 4.

After the image modification is done, the

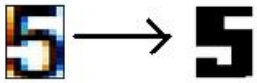

Figure 4: Figure 4. Number modification.

Number is compared to base images depicting numbers from 0-9, which had been edited in a similar process. The matching pixel count (if it's black on the number, is it black on the base image?) is counted. However, this yields an issue; because there are so many pixels and the overlap is so great, the accuracy rate is in wanting. For example, in Figure 5,

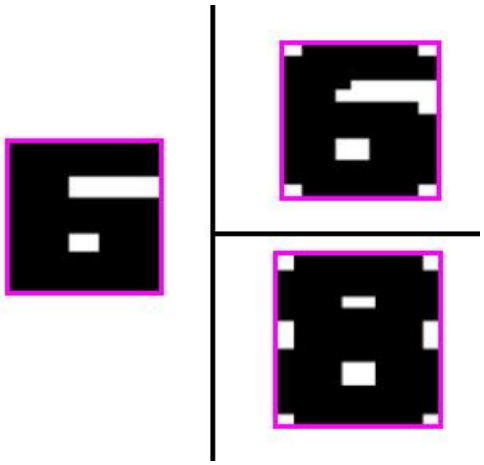

Figure 5: Figure 5. Number Comparison. The Number is on the left; the base images are on the right.

the Number is a 6. However, the compared 6 and the compared 8 are not that much different. It also should be noted that the font is different, so that the expansion of the base numbers yielded curved edges, whereas the Number 6's expansion did not. These little differences threw off the general comparison.

3.4 Scale Compilation

While this was never coded, scale compilation would have been the last step in this project. It would involve taking the Numbers and what the computer represented their quantitative value to be, and then using deductive reasoning to figure out the scale.

To determine the scale, all that is needed are two Numbers along the same axis correctly identified. For example, if one Number was 3 and another was 1, and there was a twenty pixel difference between the two, I could infer that the scale is one unit per every ten pixels.

This logic can also be used to check the inaccuracies of the image analysis of the previous section. Many differences between many different Numbers can be taken, and the rate found; the mode unit / pixel scale would be the scale itself.

The problems with this, however, are how to determine where exactly the Number lies in accordance with the axis. It is possible to use the center of the number; it is also possible to use tickmarks, if they are there. However, this is a rather arduous task.

4 Analysis

All analysis is done visually; the GUI will correctly display whatever is highlighted. It is the simplest way to determine whether or not the code is working; if the incorrect items are highlighted, then it is cause for going back and changing the code.

The GUI itself focuses on demonstrating what can be done with the project. It can display the edges, the "extended" image, and can also identify the axes / numbers / graph points. Radio buttons allow for the viewing of the aforementioned objects.

When another graph, created by Microsoft Excel, was used as a new input, the results yielded for the current code proved to be fairly consistent with what was desired; the identify and extend methods worked effectively. However, there was a slight error in the orders in which the numbers along the scale were read. Because the method places the subimages of numbers into the array according x-coordinates and then y-coordinates, it is generally assumed that the generated graphs all have the same minimum x-value.

5 Strengths and Weaknesses

While the program works in the basic principles, more efficient methods of recognition are in wanting in case graphs could become more advanced. Many assumptiosn are being made in order for this program to work; while the assumptions whill most likely hold true for all graphs (that axes are the longest chain of connected points, that edge analysis will be so simple against a white background), it is not safe to assume that it's applicable for all situations. However, the program is very effective at its purpose – showing a step by step method of how to analyze a graph, showing the viewer first the edges, then splitting them into recognizable categories, then analyzing the numbers and comparing.

6 Future Study

Areas for future study include expanding the edge detection algorithms and for improving the image analysis. Also incorporating part four of the development, the scale compilation. Much of this project can afford to be fully fleshed out, and different classes be developed. There is much in the field of image analysis that can be tried.

7 Credits

Many thanks given to Mr. Latimer, who kept us on track even when we wandered so far from it that we lost sight of passing trains.

References

[1] Alagoz, "Obtaining Depth Maps From Color Images by Region Based Stereo

Matching Algorithms", *OncuBilim Algorithm And Systems Labs, Vol. 08*, article 4, 2008.

[2] Faure and Vincent, "Document Image Analysis for Active Reading", *ACM International Conference Proceeding Series, Vol. 259.*, pp. 7 - 14. 2007.

[3] J. Fang, S. Fang, Huang, and Tuceryan, "Digital geometry image analysis for medical diagnosis", *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 217 - 221, 2006.

[4] Hartley, Catalyurek, Ruiz, Igual, Mayo, and Ujaldon, "Biomedical image analysis on a cooperative cluster of GPUs and multicores", *Proceedings of the 22nd annual international conference on Supercomputing*, pp. 15 - 25, 2008.

[5] Igathinathine, Pordesimo, Columbus, Batchelor, and Methuku, "Shape identification and particles size distribution from basic shape parameters using ImageJ", *Computer and Electronics and Agriculture*, pp. 168-182, Fall, 2008.

[6] Tasdemir, Yakar, Urkmez, and Inal, "Determination of body measurements of a cow by image analysis", *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, Article 70, 2008.