Neil Satra

Sketching Charts

Computer Science Tripos, Part II

Pembroke College

May 7, 2014

Proforma

Name:	Neil Satra
College:	Pembroke College
Project Title:	Sketching Charts
Examination:	Computer Science Tripos, Part II, 2014
Word Count:	TODO
Project Originator:	Alan Blackwell, Neil Satra
Supervisor:	Alistair Stead

Original Aims

This project aims to explore if users are able to create content faster, or experiment with it more, if given the tools to directly manipulate their creation. Specifically, it involved:

1. Building an application that lets users create graphical visualisations of their data by simply sketching their desired output, like they would on paper.

2. Evaluating the learnability of the interface, and how it compares to existing tools for creating charts, through a user study.

Work Completed

I have completed all core work items by building a Chart component for desktop applications that successfully runs sketch recognition on user input to determine what they intend to create, and then generates those chart graphics for them. I have also designed and built an interface that makes the learning curve for interacting with this component short and gentle, by applying HCI principles.

A few extensions were also completed, including allowing edits to the formal chart to flow back and manipulate the raw sketches.

Special Difficulties

None.

Declaration of Originality

I, Neil Satra of Pembroke College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date

Contents

Chapter 1

Introduction

1.1 Motivation

This project is an exploration of Human Computer Interface concepts governing the interactions of users with tools that let them explore and visualise data.

The design of currently available charting tools were constrained by the input devices available previously: mouse and keyboard. Thus, they usually allow graph generation through one of two interfaces:

1. A series of dialog boxes and wizards to walk the user through a number of choices.

2. Writing code that is interpreted to process data and generate graphics.

Because these designs were constrained, they cannot offer certain benefits, such as easier learnability and direct manipulation.

1.2 Project Description

This paper describes a different interface, which allows the user to sketch a subset of a chart on their computer touch screen like they would on paper. The hypotheses are that:

H1 This interface is more 'learnable' over time

H2 It allows easier modification and exploration of visualisations compared to other charting applications.

Both these hypotheses were investigated through a user study.

The end result is a proof-of-concept charting application that works as below:

1. The user imports data from a Microsoft Excel file.
2. They sketch a rough indication of a chart.
3. They drag the data onto elements of the chart to actually bind the data to the chart.
4. The tool then creates a 'formal' chart.
5. The tool transforms the user's original sketch to more closely match the formal chart, making the mapping between sketch and formal chart elements evident to the user.
6. Any changes on either the sketch or formal chart is fed through to the other view. For example, erasing the a sketched bar removes a data series from the formal bar.

1.3 Background and Related Work

Sketching inputs have been studied since the 1960s (Sutherland, 1964) as more natural interfaces to computers, especially for graphics-related tasks. This has largely been motivated by the widely recognised importance of interaction to Information Visualisation (InfoVis) (Lee and Isenberg, 2012). Additionally, the metaphor of sketching on paper can encourage exploratory work due to the ease of creating changes and visually expressing what sort of change one is trying to make (**?**), minimising the gap between a person's intent and the execution of the intent.

Meanwhile, there has been increasing adoption of touch-enabled phones and multi-touch slates amongst the general public, demonstrating people's affection for what have been referred to as Natural User Interfaces Lee and Isenberg (2012).

Chapter 2

Preparation

This project involved vast exploratory design work, in preparation of the actual implementation.

2.1 Requirements

This project could go in numerous different design directions from the start. Since the functionality and its benefits over existing systems depended heavily on the design chosen, it was hard to separate what benefits the program would achieve from what features the program would include and how it would expose them to the user.

However, in order to focus our exploration and make sure that the focus was on achieving some real deliverables for end users, a requirement analysis was necessary. The following functional goals were listed based on the project proposal, which focus on what basic tasks the system must help the user achieve, while allowing leeway in how the program exposed and implemented these features.

The system must allow the user to:

Core 1: Use any data they have in reasonably arranged formats in common file types.

Core 2: Specify the type of chart they want by drawing a likeness of it on screen using a stylus.

Core 3: Bind the data to the chart using an interface that makes it clear how the data is affecting the visualisation.

Core 4: Specify visual, size and positioning properties of the chart through the sketches.

Core 5: Manipulate the visual appearance of the created chart.

In addition, time permitting, the system may:

Extension 1: Transform user-drawn sketches to show the visual link to the formal chart elements.

Extension 2: Feed back manipulations applied to the formal layer back to the user drawn sketches, in order to keep the visuals of the formal and sketch layers in synchronisation.

Extension 3: Allow users to undo actions by erasing sketches, and remove the corresponding formal elements without throwing errors.

Extension 4: Allow users to manipulate any property of chart elements, not just one, so that the domain of visualisations they can create is infinite. For example, allow them to bind not just the height of bars in a bar chart to data, but also their width and colour.

Extension 5: Analyse the data and infer properties that may allow it to automatically suggest properties of the chart, such as which field belongs on which axis, or whether a data series should be log scale or linear scale.

Extension 6: The user must be able to export the chart as a Microsoft Chart object that can be embedded as a dynamic object in Microsoft Office files, not just as a raster image.

The core of this project focusses on making more usable software, rather than providing additional functionality, compared to existing systems. Hence, some usability goals were also specified:

Usability 1: Users must be able to create charts at least as quickly as they can using current systems.

Usability 2: Users must be able to build a mental model of the software's behaviour within 2 uses of it. They should thus be able to accurately predict the consequences of any action taken within the software.

Usability 3: Changes to the information visualisation must occur through directly manipulating the visual representation of the chart, rather than through disconnected User Interface widgets.

Usability 4: The user must be able to easily try out changes to the visualisation, see what that would look like, and undo them if needed.

2.2 Work Items

An iterative development process, similar to the Spiral Model was adopted for this project. This allowed early experimentation with and user testing of the various components and different interface designs. The following broad work items were identified as necessary to achieve the objectives above:

1. Assess the various methods to build a classifier for ink recognition, including using the RATA Framework
2. Run an initial user study to see how people naturally draw graphs, and also use it to collect training examples for the classifier.
3. Build a UI that accepts strokes, runs them through the classifier, and shows the user feedback to indicate successful recognition.
4. Build the UI widget that lets users import their spreadsheets in Microsoft Excel (xlsx) or Comma Separated Value (csv) files. It must then expose the various fields detected.
5. Build the charting component to convert the recognised sketches and the ink into a finished visualisation.
6. Run a pilot study followed by a user study to evaluate the system.

2.3 Development Environment

For this project, the hardware available was a Microsoft Surface Pro (1st gen), which features the active digitizer screen required for precise inking. Since this machine runs Windows by default, we chose to develop the system using the .NET framework, which has built-in support for Tablet PCs and Ink handling.

As a precaution, insurance was taken out on the machine to ensure quick replacement in case of damage or loss. Additionally, version control was used extensively in the project, to ensure no work was lost. The code for the RATA ink stroke recogniser, described below, was uploaded to a Git repository in collaboration with RATA's authors. The code for this project was then written in a fork of that repository, to allow updates to RATA to be pulled in. The dissertation itself was written in LaTeX, so that the text files could be versioned in another Git repository. Both repositories were backed up off-site on repository host Bitbucket. The dissertation was also backed up online using file synchronisation software Microsoft OneDrive.

2.4 Building the classifier

Core to the system is an ink recognition component that identifies the chart element (e.g. bar, axis) that the user has sketched. This must work above a certain accuracy threshold or the system will prove frustrating to users (Frankish et al., 1995). However, since the project scope included other components too, the time available to build this classifier was limited. Hence, we decided to build a classifier using existing tools rather than implementing one from scratch.

2.4.1 Tools used

The digital ink library Microsoft provides includes an ink stroke recognizer. There are a number of other ink recognition tools available, such as $1, Rubine, PaleoSketch and CALI. However, while these recognizers work well for the recognition tasks they were designed for, the RATA framework is explicitly designed to allow generation of a custom recogniser for a new domain using a few example images (Chang et al., 2010). It thus outperforms a number of these recognisers on data sets other than the examples they ship with. By choosing to use RATA, we ensured we got a high accuracy rate for a domain of chart elements that may need to expand over time to incorporate more types of charts.

Rather than using manually specified features and thresholds for these features, RATA uses machine learning to automatically select features and identify relationships between them. It uses the WEKA tool, which provides numerous different data mining algorithms, and combines the results of these

algorithms to provide higher accuracy. The authors of RATA also provide a 'Data Manager' tool to facilitate easy collection and labelling of training data.

2.4.2 Data collection

After acquiring RATA, we inspected the code and did manual testing to find some blocking bugs. Since we were in contact with the authors of the software, including Beryl Plimmer who until recently worked at the University of Cambridge Computer Lab, we were able to confirm with them that these were indeed bugs. We implemented fixes for them and contributed them back to the authors, and are working towards getting the code ready for to be published Open Source.

With a working version of RATA, an initial study was run to collect training data. We asked 10 participants (20-26 year old Cambridge students, studying a large variety of subjects) to draw a chart. I spoke out the following prompt:

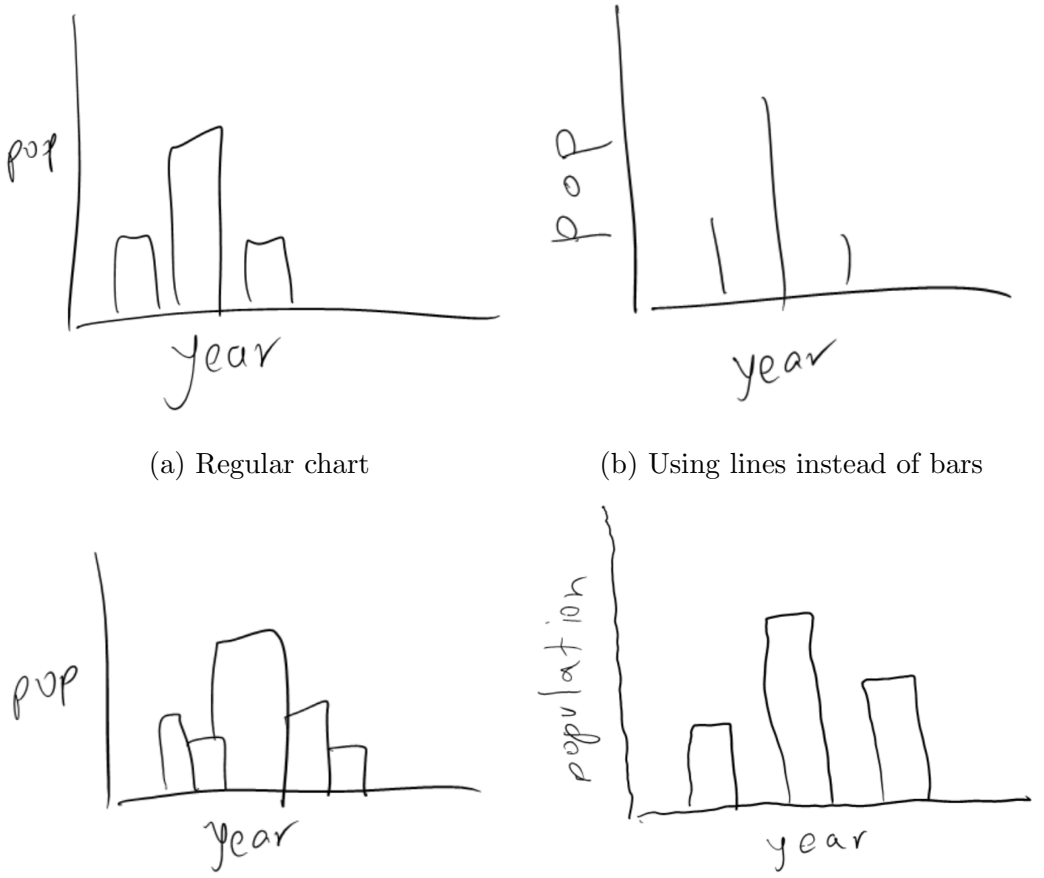
> Imagine you are a government official trying to use a bar chart to visualise how the population has grown over time. Can you sketch out what this bar chart might look like? Just treat this screen like paper.

They were then presented a simple UI with a large white canvas and the following task description written in a panel:

> Draw 2 axes. Label the x axis 'Year' and the y 'Population'. Draw 3 bars of different heights. Each shape (axis, bar) should be drawn in one stroke.

They were asked to draw the same chart 2 times, in order to get 20 training samples in total, and to observe how much variation there is between multiple sketches by the same user. On the second drawing, they were encouraged to draw a less conventional chart, to make the system as robust in the face of variations as possible.

Three elements were then defined: Axis, Bar and Text (an extra element, 'L Axis' was added later based on feedback from a pilot user study). We went through each figure and labelled the various elements.



(a) Regular chart

(b) Using lines instead of bars

(c) Grouped bars

(d) Using a mouse instead of the stylus

Figure 2.1: Some of the more unusual chart sketches collected

Once all the elements in all the charts were labelled, we could begin training. RATA includes a dataset generator tool that allowed easy extraction of various features of the strokes, such as 'distance from first to last point', 'absolute curve of largest segment' and 'pressure variation'. Data for 121 such attributes, about 270 ink strokes was compiled into a .csv file for use in training.

2.4.3 Training

The labelled data was then sent to a 'Vote' classifier in Weka, which combines the probability distributions derived from multiple classifiers. Specifically,

Figure 2.2: Elements of the sketch labelled as Axis (cyan), Bar (brown) or Text (dark blue)

the types of classifiers combined were Logit Boost, Bayes Net, LMT (Logistic Model Trees) and Random Forest. In order to assess how well each of these individual classifiers were performing, an experiment was set up using Weka Experimenter. The data collected in the initial study was shuffled, and then 66% was chosen randomly as training data, the rest as testing. Then a paired T Test gave the following results:

Tester: Paired Corrected T Tester
Analysing: Percent correct
Confidence: 0.05 (two tailed)

Table 2.1: Classifier Algorithms Used

(1) meta.LogitBoost '-P 100 -F 0 -R 1 -L -1.7976931348623157E308 -H 1.0 -S 1 -I 10 -W trees.DecisionStump' 8627452775249625582
(2) bayes.BayesNet '-D -Q bayes.net.search.local.TAN – -S BAYES -E bayes.net.estimate.SimpleEstimator – -A 0.5' 746037443258775954
(3) trees.LMT '-P -I 50 -M 200 -W 0.0 -A' -1113212459618104943
(4) trees.RandomForest '-I 100 -K 0 -S 1' -2260823972777004705

Table 2.2: Results

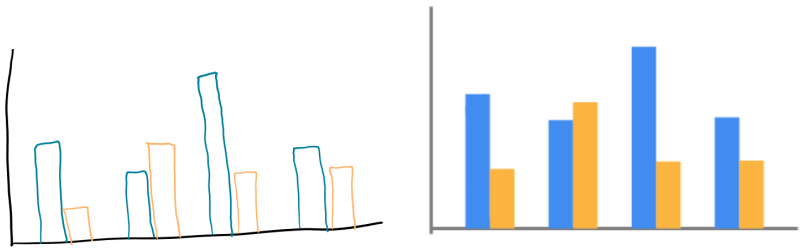
Dataset	(1)	(2)	(3)	(4)
'Initial study'	97.05	98.15	98.80	96.07

○, ● statistically significant improvement or degradation

2.5 Design

Now that we had shown that a classifier could be built with relatively small sets of training data that still performed well, we had to design the interface to expose this functionality. This largely involved assessing tradeoffs between choices.

2.5.1 'Sketchy' or Formal



(a) Sketchy look (b) Formal look

User content can be in shown in two styles: sketch or formal. (Yeung et al., 1990) showed that a doodle-like appearance encourages early-stage experimentation and discussion, whereas a formal appearance looks finished and professional. Thus, we had to choose whether the system should generate the rest of the chart with a 'sketchy' look, or convert the input into a finished chart with a formal look. Generating the 'sketchy' look can be non-trivial, (Plimmer et al., 2010; Wang et al., 2011) e.g. if the user draws one bar, how should the system generate more bars that look hand-drawn by the same author but not just like stretched versions of the first bar? Additionally, if the users want to present these charts to an audience, they must look polished, and so at some point an export option for a formal look needs to be offered. Thus, we decided to offer a hybrid that shows the user's input in sketch form, but also the formal chart generated by the system.

2.5.2 Modes or modeless

Since the application now has both sketch and formal content, the user must have an easy way to switch between the two views. One approach is to give the user an explicit UI widget to toggle between the two modes. This way, the user explicitly indicates what they want to see, and thus should have a

better understanding of what state the system is in. This also allows the system a chance to change the controls available to the user.

The other approach is to avoid modes, requiring lesser cognitive effort from the user since they don't need to keep track of what state the system is in. In this project, this could have been done by showing the sketch view when the user was about to edit the chart. The active digitizer hardware allows the system to detect when the user brings the stylus within range of the screen, just before they actually touch the screen, allowing the chart to be in sketch view by the time the stylus is down. When the stylus goes out of range, the system can switch to the formal chart view. This would solidify the metaphor that edits are done to the sketch, but the final product to be looked at is the formal view.

At first, the modeless version was chosen for its lower cognitive overhead. However, when the extension to allow edits not just to the sketch view, but also the formal view, was undertaken, we had to switch over to a mode-based system to allow the user to interact with the graph in both views with their stylus.

2.5.3 Standard or custom charting

Since the system is generating a formal version of the chart, a charting component is required to render this visualisation. The .NET framework comes with built-in chart controls that offer basic functionality with relatively low implementation effort. They also allow easy export as dynamic chart objects into Microsoft Office files. However, customising their appearance beyond a certain point is extremely difficult, making it easier to just make one's own charting component from scratch and control all aspects of the rendering. This means having to re-implement a lot of core functionality though, such as scaling shapes correctly, choosing labels that are round figures when possible, and generating colours than work well together for different data series. This also means that the chart can only be exported as a raster image rather than as a chart object.

To enable rapid prototyping, we chose to utilise the standard charting component at first. As our needs to customise the chart grew, we were able to make our own chart class that implemented the same interface as the standard component, and so could be slotted in to replace it.

2.5.4 Finite or infinite domain

Some tools, such as Microsoft Excel, let the user make one of a limited set of charts, such as bar or pie charts, quickly. Others, (which usually involve coding), such as D3.js, let the user make a vast variety of visualisations by creatively combining basic elements like lines, boxes and wedges. However, these require expert knowledge of the tools, and take longer to even create basic visualisations.

Library of charts	**Modular charts**
Draw basic gestures or elements to indicate which chart type is desired	Draw any one of 7 basic components (lines, bars, labels etc) and bind data to attributes of theirs such as width, height, colour or radius
Finite domain	Infinite domain
Quick	Slower
Simple interface, just drop data on an element to bind data	Complex interface to expose all attributes and manage data binding

While an infinite domain system would have been intriguing to explore, it would contradict the project's primary usability goal that the system should be faster than users' current systems. Additionally, the 80/20 rule indicated that while a few power users may want to generate custom visualisations, the majority would just want to make simple charts. Thus, the additional functionality didn't justify the additional complexity for the average user.

Chapter 3

Implementation

3.1 Overview

At a high level, the program is composed of 3 components - data handling, sketch processing and charting (in increasing order of complexity). It is written in an Object Oriented fashion, with separation between the views (Windows Forms) and controllers (C# classes), to allow for easy testing.

3.2 Data import and management

Since the application is targeted at the average user, their data is most likely to be stored in spreadsheet format. Thus, it is important to allow them to import data from .xlsx and .csv files. For the sake of simplicity, the code assumes that the data is well-formed. Specifically, it works on the following assumptions:

1. The data is arranged as records in the rows of the spreadsheet.
2. The first row contains the names of the various fields.
3. No data is missing (if there are m columns and n rows, there are $m \cdot n$ data values.

Under these assumptions, importing tabular data is a common use case, so I studied a number of existing libraries and methods to do this in C# and ultimately settled on built-in OLE data import functionality.

3.3 Sketch Processing Workflow

3.4 Charting

Bibliography

Chang, S., Plimmer, B., and Blagojevic, R. (2010). Rata. ssr: Data mining for pertinent stroke recognizers. *Proceedings of the Seventh ....*

Frankish, C., Hull, R., and Morgan, P. (1995). Recognition accuracy and user acceptance of pen interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*, pages 503–510.

Lee, B. and Isenberg, P. (2012). Beyond mouse and keyboard: Expanding design considerations for information visualization interactions. *Visualization and ...*, (October).

Plimmer, B., Purchase, H. C., and Yang, H. Y. (2010). SketchNode : Intelligent sketching support and formal diagramming. pages 136–143.

Sutherland, I. E. (1964). Sketchpad a Man-Machine Graphical Communication System.

Wang, M., Plimmer, B., Schmieder, P., Stapleton, G., Rodgers, P., and Delaney, A. (2011). SketchSet: Creating Euler diagrams using pen or mouse. *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 75–82.

Yeung, L., Plimmer, B., Lobb, B., and Elliffe, D. (1990). Effect of Fidelity in Diagram Presentation.