LEARNING DESIGN: AESTHETIC MODELS FOR COLOR, LAYOUT, AND TYPOGRAPHY

by

Peter O’Donovan

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Copyright © 2015 by Peter O’Donovan
Abstract

Learning Design: Aesthetic Models for Color, Layout, and Typography

Peter O’Donovan
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2015

This thesis presents aesthetic models for three key components of graphic design: color compatibility, layout creation, and font selection. We develop these models using a variety of techniques, including machine learning and data mining, crowdsourcing, and optimization, and demonstrate their use in practical design applications. We first model color compatibility from large datasets, including analysis of existing color theories, a learned model for predicting compatibility, and several applications using this model. We then use a collaborative filtering approach to model color preferences for individual users. Secondly, we present an energy-based model for single-page design layouts derived from design principles, using novel analysis algorithms. We show how this model can be trained using existing layouts, and demonstrate its use on various tasks including style-based synthesis and retargeting. We then extend this model to enable an interactive, suggestion-based layout tool. Finally, we model fonts using attributes and similarity data from crowdsourced studies, then use these models to develop exploratory font selection interfaces.
Acknowledgements

I dedicate this thesis to the following:
To my teachers, for their wisdom, guidance, and patience.
To my fellow students, for their advice, inspiration, and commiseration.
To my family, for their supportive phone calls, home-cooked meals, late-night films, and love.
To my friends, and the countless hours together, that made these years the happiest of my life.

Firstly, I am indebted to both my supervisors Aaron Hertzmann and Aseem Agarwala, who collaborated with me closely for all this work. The core ideas of this thesis were all born from our many discussions over the years. Chapter 7 was joint work with Jānis Libeks, who was instrumental in realizing the project. Many thanks are also due to my thesis committee members, David J. Fleet, Karan Singh, and to my external examiner Craig Kaplan for their criticism and support throughout this process and for their specific help improving this document.

I am very thankful for the financial support provided by the National Sciences and Engineering Research Council of Canada (NSERC) and the University of Toronto’s Computer Science Department.

I would also like to thank John Hancock and Ingrid Varga for the many hours they spent building and supporting the varied architecture required for this work.
Contents

Acknowledgements iii

1 Introduction 1
  1.1 Thesis Statement ................................................. 3
  1.2 Overview and Contributions ................................... 4

2 Related Work 8
  2.1 Theories of Aesthetics ......................................... 8
  2.2 Color Aesthetics ............................................... 9
  2.3 Image Aesthetics ................................................. 12
  2.4 Individual Aesthetic Preferences ............................. 13
  2.5 Design Layout Aesthetics .................................... 14
  2.6 Design Interfaces .............................................. 17

3 Color Compatibility from Large Datasets 19
  3.1 Datasets .......................................................... 21
  3.2 Model-Free Data Analysis ..................................... 23
    3.2.1 Distribution of Themes .................................... 23
    3.2.2 Preferred Colours and Color Pairs ...................... 25
    3.2.3 Hue Templates ............................................. 27
    3.2.4 Hue Entropy ................................................ 30
    3.2.5 Theme Names ............................................... 32
  3.3 Learning Color Compatibility ................................. 33
  3.4 Model-Based Data Analysis .................................... 37
    3.4.1 Important Features ........................................ 37
    3.4.2 Ratings Across Datasets ................................. 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.7</td>
<td>Flow</td>
<td>87</td>
</tr>
<tr>
<td>5.4.8</td>
<td>Unity</td>
<td>88</td>
</tr>
<tr>
<td>5.4.9</td>
<td>Previous Layout</td>
<td>89</td>
</tr>
<tr>
<td>5.5</td>
<td>Optimization</td>
<td>90</td>
</tr>
<tr>
<td>5.6</td>
<td>Learning Model Parameters</td>
<td>91</td>
</tr>
<tr>
<td>5.7</td>
<td>Applications</td>
<td>92</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Energy Terms and Score Correlation</td>
<td>102</td>
</tr>
<tr>
<td>5.8</td>
<td>Discussion</td>
<td>104</td>
</tr>
<tr>
<td>6</td>
<td>Graphic Design with Automatic Layout Suggestions</td>
<td>105</td>
</tr>
<tr>
<td>6.1</td>
<td>User Experience</td>
<td>106</td>
</tr>
<tr>
<td>6.2</td>
<td>User Interactions</td>
<td>108</td>
</tr>
<tr>
<td>6.3</td>
<td>Design Model</td>
<td>110</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Energy Function</td>
<td>110</td>
</tr>
<tr>
<td>6.3.2</td>
<td>User Constraints</td>
<td>111</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Model Details</td>
<td>112</td>
</tr>
<tr>
<td>6.4</td>
<td>Synthesizing Layouts</td>
<td>114</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Optimization</td>
<td>114</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Style Sampling</td>
<td>116</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation</td>
<td>118</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Design Quality Study</td>
<td>118</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Preference Study</td>
<td>119</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Qualitative Study</td>
<td>122</td>
</tr>
<tr>
<td>6.6</td>
<td>Discussion</td>
<td>123</td>
</tr>
<tr>
<td>7</td>
<td>Exploratory Font Selection Using Crowdsourced Attributes</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>Related Work</td>
<td>127</td>
</tr>
<tr>
<td>7.2</td>
<td>User Experience</td>
<td>129</td>
</tr>
<tr>
<td>7.3</td>
<td>Estimating and Predicting Font Attributes</td>
<td>131</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Font Selection</td>
<td>132</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Attribute Selection</td>
<td>132</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Attribute Estimation</td>
<td>133</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Attribute Prediction</td>
<td>135</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Model Evaluation</td>
<td>137</td>
</tr>
</tbody>
</table>
7.4 Font Distance Metric ...................................................... 140
  7.4.1 Metric Learning ...................................................... 140
  7.4.2 Grouping ............................................................. 143
7.5 User Interface Evaluation ................................................. 144
  7.5.1 Font Matching Task ................................................. 144
  7.5.2 Design Task .......................................................... 146
7.6 Discussion ................................................................. 151

8 Conclusions ................................................................. 152

Bibliography ................................................................. 159

Appendix A: Color Theme Dataset Details ............................... 174
Appendix B: Theme-Template Distance .................................. 175
Appendix C: Optimization Proposals ...................................... 175
Appendix D: Font Features .................................................. 178
Appendix E: Font MTurk Study Details ................................. 179
1

Introduction

When first my eyes saw thee,
I found me thy thrall,
By magical drawings,
Sweet tyrant of all!

– Ralph Waldo Emerson, Ode to Beauty

In the modern world, graphic design is everywhere. We’re inundated by designs in cereal boxes, smartphone apps, road signs, book covers, posters, logos, magazines layouts, banner ads, fonts, web pages, among dozens of others. Design also stretches across the globe, even into remote villages of developing nations. Visually communicating information is crucial for countless tasks, and creating these designs requires a huge number of professionals. There are over 250,000 professional graphic designers working in the USA alone [114]. On top of this, there are millions of amateur designers who routinely engage in design tasks like creating presentation slides, websites, or even party invitations. How strange and exciting then, that this area of computer graphics, with millions of tool users, affecting the lives of billions of people, remains so poorly understood.

There are many good reasons why graphic design is hard to study. The breadth of design makes it difficult to build comprehensive models. Designs can range from graphical to textual, from the purely aesthetic to the purely functional, from a single brush stroke to an entire web site. Ironically, while design is ubiquitous, it is also hard to find datasets of designs. Many designs are copyrighted, commercial works created by professionals, and are not shared publicly. Lastly, the key goal of most graphic designs is visually communicating specific information. To build models of graphic design, we should understand how people perceive and process visual stimuli,
a difficult task which vision researchers have been deeply exploring for half a century.

Given the breadth of graphic design, there are many interfaces for different domains, ranging in complexity from simple template-based tools like PowerPoint, to byzantine tools like Illustrator. Unfortunately, simple design tools are often restrictive, and deviating from templates is time-consuming and tedious. While complex tools permit a great deal of control, they are difficult to learn and provide little automatic assistance. Design is an artistic and exploratory process, and ideally one that should be fun! Unfortunately, existing tools hinder the artistic process in many ways, and are often quite painful for novices. Design is often about exploring alternatives, but current tools provide few suggestions, and force users into a serial mode of refining a single design. For example, with current tools choosing colours involves simple color pickers, and provides no help for choosing compatible colours. Even choosing a font can be intimidating, with the user scrolling through hundreds of fonts with meaningless names like Caslon, Helvetica, or Architect’s Daughter.

To build more sophisticated tools, however, we require a deeper understanding of how people perceive designs. To help create layouts, tools should model how people read layouts, which elements are considered the most important, how groups of elements are perceived, etc. These perceptual properties, such as image saliency[115], are important for building design tools that can make automatic suggestions. For example, it’s often crucial that the design title is perceived as the most important element, and draws the eye of the viewer. Perceptual analysis of graphic designs is a relatively unexplored area of computer vision, but one that has the potential for significant impact.

Understanding art and design has long interested many philosophers, artists, and architects. In particular, the concept of beauty in design, often referred to as aesthetics, has been debated for several millennia. Until the 20th century however, studying aesthetics usually meant analytical descriptions, or proposing theories which were difficult to validate. For example, the golden mean is commonly claimed to be an aesthetic ideal in art and architecture, dating back to the ancient Greeks. However, commonly given ancient examples are incorrect and misleading [91], and empirical studies over the last century have not produced demonstrative proof of its appeal [40]. Another common heuristic is the Rule of Thirds: that elements should be placed near the “Third lines” of an image. However, work by Palmer et al. [120] has shown the Rule of Thirds to be inaccurate; people prefer objects which face the image center and symmetric objects are more aesthetically pleasing in the image center. As another example, color theorists like Goethe have suggested that colors opposite on the color wheel are complementary and should be paired together, though again, there is little empirical evidence supporting this heuristic. These aes-
thetic rules are widely reproduced in art and design texts [73, 52, 67]; empirical research into their effectiveness, or lack thereof, could benefit a wide range of artists and educators.

Over the last few decades, researchers in many fields have begun investigating aesthetics. Psychologists have conducted numerous studies into aesthetics, including evaluations of classical rules. Understanding the perceptual mechanisms of aesthetics is difficult, but recent neuroscience research has begun to offer insight into why some visual stimuli are considered more beautiful than others. For example, contrast is pleasing in art partly since our visual systems are designed to perform object detection, where the contrast between objects is crucial [126]. Graphics researchers have also begun to simulate artistic styles like paintings, line drawings, or technical illustration, enabling hypothesis testing on how these objects are created and perceived. However, graphic design has received relatively less interest than other artistic fields, and many components of design, such as color and layout, remain poorly understood.

Recently, the explosion of web data has allowed researchers to perform large-scale data analysis and data-driven learning of aesthetic models. The websites Kuler and COLOURlovers have millions of color themes, i.e., ordered combinations of 1-5 colors, created by thousands of users, and sites like Flickr and Photo.net include billions of images, including many graphic designs. Online crowdsourcing sites, like Amazon Mechanical Turk, can also be used to perform perception studies for a fraction of the time and cost of traditional studies, allowing researchers to more easily test new aesthetic theories and tools.

There has also been important recent work in the field of machine learning, which is highly relevant for building computational models of design. Some of these advances include optimization-based approaches for parameter estimation [82], collaborative filtering techniques to predict user preferences [146], and estimating relative scalar attributes for images [122].

Combining new web data, crowdsourcing approaches, and machine learning techniques, presents an unprecedented opportunity to analyze how design and aesthetics work in practice. Given these new techniques, it is now much simpler to investigate how people perceive designs, develop models, and finally, build practical tools to help designers.

1.1 Thesis Statement

Our thesis is the following:

The breadth and complexity of graphic design encourages a decompositional approach, with deeper investigations into the individual components of design. A
wide variety of techniques, including data-mining and machine-learning, crowdsourcing, and optimization, are helpful for building these component models, which can then be used to understand aesthetics, and develop tools to help designers.

Given the complexity of graphic design, it is difficult to build general models which cover the huge range of possible designs. In this work, we attempt to decompose the problem of design modelling, and concentrate on key sub-problems of design aesthetics: color, layout, and typography. For these areas, we use a variety of approaches to build a model for understanding a visual phenomenon, and then use the model in practical applications or interfaces.

We first investigate color theme aesthetics since it presents a simple domain for initial tests in aesthetic modelling. We use large online datasets and crowdsourced studies to train a simple linear regression model to predict color compatibility. We use linear regression due to its good performance, and since the learned weights allow analysis of the importance of different features. We then explore applications including extracting themes from images, and colouring graphic designs. Collaborative filtering approaches are also used to model individual user preferences and theme similarity.

Design layouts are more complicated than color themes however, and require a more complex synthesis model. We therefore develop an energy-based layout model for single-page graphical design such as advertisements, fliers, or posters. This model includes energy terms derived from design principles, such as alignment and balance. We train the model using example layouts, then show applications including generating design layouts in various styles and retargeting designs to new sizes. We further demonstrate the model’s usefulness by building a layout creation interface which provides automatic layout suggestions.

Finally, we turn to typography and font selection. We model fonts using descriptive attributes, such as “fun” or “legible,” as well as using font similarity. For both attributes and similarity, we gather data using crowdsourced studies, and train models which can predict attributes and similarity for new unseen fonts. These models are incorporated into exploratory font selection interfaces, allowing designers to more easily navigate the huge space of possible fonts.

1.2 Overview and Contributions

We begin by a discussion of related work in Chapter 2, then discuss our aesthetics models in turn:
1.2. Overview and Contributions

**Color Compatibility From Large Datasets.** In Chapter 3, we study color aesthetics and compatibility theories using large datasets, and develop new tools for choosing colors. There are three parts to this chapter. First, using online datasets, we test new and existing theories of human color preferences. For example, we test whether certain hues or hue templates may be preferred by viewers. This chapter includes the first large-scale online studies of color preference, the first studies based on five color themes, and the first studies of user-generated color combinations. Second, we present the first large-scale data-driven modelling of color preferences. Specifically, we learn quantitative models that score the quality of a five color theme. Third, we demonstrate simple prototypes that apply a learned model to tasks in color design, including improving existing themes, extracting themes from images, and colouring graphic designs.

The project page is at [www.dgp.toronto.edu/~donovan/color](http://www.dgp.toronto.edu/~donovan/color). This page contains the datasets used in this chapter, as well as code for training and testing the compatibility model.

**Collaborative Filtering of Color Aesthetics.** In Chapter 4, we extend the work of the previous chapter to model individual user preferences for color themes. We investigate a collaborative filtering approach to model preferences, and use matrix factorization to learn latent vectors for users and color themes. We also propose two extensions to the probabilistic matrix factorization framework. We first describe a feature-based model using learned transformations from feature vectors to a latent space, then extend this model to nonlinear transformations using a neural network. We find that using features improves performance for predicting color aesthetics. We also find that modelling individual user preferences outperforms an average aesthetic model which ignores personal variation. Lastly, we use the learned latent vectors for measuring theme similarity and visualizing the space of color themes. More generally, this work is among the first to learn models of individual preference in visual aesthetics.

The project page is at [www.dgp.toronto.edu/~donovan/cfcolor](http://www.dgp.toronto.edu/~donovan/cfcolor), and contains code and datasets used in this chapter.

**Learning Layouts for Single-Page Graphic Designs.** In Chapter 5, we present an approach for automatically creating graphic design layouts using a new energy-based model derived from design principles. The model includes several new algorithms for analyzing graphic designs, including the prediction of perceived importance, alignment detection, and hierarchical segmentation. Given the model, we use optimization to synthesize new layouts for a variety of single-page graphic designs. Model parameters are learned with Nonlinear Inverse Optimization (NIO) from a small number of example layouts. To demonstrate our approach, we show
results for applications including generating design layouts in various styles, retargeting designs to new sizes, and improving existing designs. We also compare our automatic results with designs created using crowdsourcing and show that our approach performs as well as, or better than, novice designers. Graphic design is a particularly hard task to automate. Even humans are not particularly good at it, apart from designers who often have significant experience and specialized training. By contrast, classic problems like speech recognition and computer vision are performed well by most humans. Automatically producing layouts on par with untrained human designers represents significant progress on this unsolved problem.

The project page is at www.dgp.toronto.edu/~donovan/layout. This page contains numerous examples to supplement the chapter, including dozens of sample importance maps, all training data for our models, and examples of designer and crowdsourced layouts.

**Graphic Design with Automatic Layout Suggestions.** In Chapter 6, we present an interactive system for creating graphic designs using automatic layout suggestions. The system adapts and simplifies the model of Chapter 5 for use on the GPU, and uses the model to make two types of suggestions: refinement suggestions which improve the current layout, and brainstorming suggestions which explore large changes in style. We also investigate two modes for interacting with suggestions. First, we investigate a suggestive mode, where suggestions are shown on the side and must be accepted. Secondly, we develop an adaptive mode where elements are moved automatically to improve the layout. The two modes are compared to a baseline without suggestions by novice users, and the quality of the resulting layouts are also evaluated. We find that both modes produces significantly better designs than the baseline on average.

The project page is at www.dgp.toronto.edu/~donovan/design, and includes a video demonstrating the system.

**Exploratory Font Selection Using Crowdsourced Attributes.** In Chapter 7, we investigate design interfaces for font selection using large collections of fonts. Existing interfaces typically list fonts in a long, alphabetically-sorted menu that can be challenging and frustrating to explore. We instead propose three interfaces for font selection. First, we organize fonts using high-level descriptive attributes, such as “dramatic” or “legible.” Second, we organize fonts in a tree-based hierarchical menu based on perceptual similarity. Third, we display fonts that are most similar to a user’s currently selected font. These tools are complementary; a user may search for “graceful” fonts, select a reasonable one, and then refine the results from a list of fonts similar to the selection. To enable these tools, we use crowdsourcing to gather font attribute data,
and then train models to predict attribute values for new fonts. We use attributes to help learn a font similarity metric using crowdsourced comparisons. We evaluate the interfaces against a conventional list interface and find that our interfaces are preferred to the baseline. Our interfaces also produce better results in two real-world tasks: finding the nearest match to a target font, and font selection for graphic designs.

The project page is at www.dgp.toronto.edu/~donovan/font, and includes a video demonstrating the interfaces, as well as code, datasets, and links to the interfaces.
2

Related Work

When you make judgements on beauty, you do not follow mere fancy, but the workings of a reasoning faculty that is inborn in the mind.

– Leon Battista Alberti

In this section, we review related work on aesthetic modelling. We begin by a short discussion of high-level theories of aesthetics from psychology, neuroscience, and computer graphics. These theories provide a context for examining more specific results in color, photography, and graphic design aesthetics. For color aesthetics, related work from empirical psychology is reviewed, as well as graphics and vision applications. We then review recent data-driven approaches to learning image aesthetics. Finally, we examine previous work in modelling design layouts, as well as design interfaces. We defer the discussion of fonts to Chapter 7, as most related work is not specifically related to aesthetics.

2.1 Theories of Aesthetics

Classical theories of aesthetics focus on two opposing approaches [127]. The objectivist tradition, dating to Greek philosophers including Plato, emphasizes object properties as the source of aesthetics, including symmetry, proportion, and order. By contrast, the subjectivist tradition, dating back to the Sophists, maintains that aesthetics is based on individual and subjective responses, i.e., ‘beauty is in the eye of the beholder.’ Recent theorists often avoid this duality however, and focus on the interaction between objects and people’s response to them. Based on a review of the empirical aesthetics literature, Reber et al. [127] propose that objects which are perceived more fluently, i.e., are easy to visually process, have a higher positive aesthetic re-
response. Other researchers have also theorized that aesthetics is related to information theoretic measures which encode the complexity of an object. For example, Rigau et al. [130] investigate measures including Shannon entropy and Kolmogorov complexity on color distributions of paintings from Mondrian, Seurat and van Gogh.

Ramachandran and Hirstein [126] propose aesthetic laws derived from neurological observations and experiments. These laws include the ‘peak-shift’ principle: that people prefer an exaggeration of form or color, such as exaggerated Indian nudes or Van Gogh’s sunflowers. Exaggerations are preferred because they amplify the neural mechanisms activated by the original object. Other proposed principles include grouping based on similarity, contrast extraction, and symmetry. Neuroscience is important for aesthetic research as it can provide explanations for aesthetic preferences. For example, studies show that people find symmetric faces more aesthetically pleasing [38], possibly explaining symmetry’s importance in photography and design.

One classical measure of aesthetics which does not fit into this visual processing paradigm is the golden mean, which states that the proportion \( \frac{1 + \sqrt{5}}{2} = 1.618033... \) is aesthetically pleasing. While the ratio was first recorded by Euclid, many ancient examples of art and architecture are offered as proof of its timeless appeal, including the Great Pyramid, the Parthenon, and even the Aeneid. However, these examples are misleading, and there is no proof that ancient architects and artists employed the golden mean [91]. Empirical studies of the golden mean date back to the 1860s; in his survey, Green [40] found mixed results, likely due to poor study methodologies, with no demonstrative proof that the golden mean has a strong aesthetic value.

Computer scientists have also begun to investigate aesthetics by simulating artistic styles. Hertzmann [46] proposes the field of non-photorealistic rendering as vital for scientific investigations of visual art. Most classic approaches to aesthetics and art history describe features of artistic styles. By contrast, computational models that generate aesthetic styles allow researchers to more easily evaluate hypothesis of how art is created and perceived.

### 2.2 Color Aesthetics

Color has intrigued philosophers since the ancient Greeks [30]. Modern color theory began with Newton, who developed a color wheel based on the additive property of light. The wheel contains three primaries: red, green, and blue, which are separated by three secondaries: yellow, cyan, and magenta, created by adding the primaries. The distance to the center specifies the saturation; at the center is white, a combination of the three primaries.

Color wheels also allow color relationships to be represented geometrically. Goethe [34]
arranged the color wheel according to physiological vision phenomena such as after-images, and proposed that compatible contrasting colors are opposite on the color wheel. Many color compatibility models developed over the following centuries similarly express compatibility as geometric relationships in some color space. However, there has been little study of the effectiveness of these models.

**Hue Templates.** One of the most popular theories of color compatibility is the notion of *hue templates*, which generalizes Goethe's theory by describing compatible colors as fixed rotations about the color wheel. Hue templates are taught in many texts on art and design [52, 67]. Itten [52] proposed that sets of 2, 3, 4, and 6 hues equidistant on the color wheel were harmonious. Templates are considered equally harmonious, and rotationally invariant along the color wheel. However, designers often treat templates as starting points, rather than strict rules [97]. One weakness of hue templates is they are defined independently of the underlying hue wheel. The color theme site Kuler uses a BYR color wheel (the “artists’ color wheel”), whereas the COLOURlovers site uses an RGB color wheel, which suggest different colors using the same template rules. Fig. 2.1 shows the templates implemented in Kuler and COLOURlovers, which cover the most popular hue templates.

More recently, Matsuda [96] proposed a color harmony model using 8 hue templates (Fig. 2.1, right) and 10 tone templates, derived from fashion questionnaires given to female students in
Japan over a nine-year period, and from color themes provided by fashion companies. This model has been used in several computer vision and graphics projects \([18, 78, 157]\). However, to our knowledge, these templates have never been rigorously evaluated.

**Color Harmony.** Numerous theories have been proposed for color harmony beyond just hue \([17, 102, 116, 104]\). While the underlying color spaces typically vary, these theories often have similar rules. Many suggest colors are harmonious if one dimension of the space (such as saturation or value) contrasts while the others remain fixed, or that colors along lines in the color space are harmonious. For example, the Munsell system suggests that colors with fixed hue and value but varying saturation are harmonious. The Ostwald system suggests that colors are harmonious with equal white or black content. These sets of colors form lines in that color space.

In recent decades, psychologists have begun controlled studies of color compatibility and preferences \([39, 119, 117, 148, 96, 121, 136, 105]\). While this work is often contradictory \([136]\), a few trends emerge: colors harmonize if they have the same hue, equal or similar color saturation values, and contrasting lightness values. The data comes from tightly-controlled laboratory experiments, which forces a small number of participants (usually less than 100), a small range of colors (usually less than 100), and a small number of combinations (usually 1-3). An exception is the Colouroid system \([105]\), derived from several large-scale experiments on sets of up to three hues, though often with a small number of stimuli (e.g., only 108 color combinations for 3 hues). By contrast, in Chapter 3, we use large datasets of 5 color themes from thousands of participants from across the globe with a wide range of colors and viewing conditions. Another significant difference is that we explore compatibility in user-generated color combinations. In addition, we also evaluate hue templates, the most widely used color harmony model.

**Color Applications.** Color themes and compatibility models have been used in a variety of recent graphics and vision work. Color harmonization \([18]\) optimizes the histogram of hues in an image to lie within the closest of Matsuda’s hue templates. Images may also be harmonized with respect to a theme from other sources, such as flags. Wang et al. \([162]\) modify an image to match a user-selected color theme, using the color mood model of Ou et al. \([118]\) to improve color compatibility. Lalonde and Efros \([72]\) use a data-driven color compatibility model to evaluate image realism for realistic recolouring and compositing. Csurka et al. \([19]\) learn associations between color themes and keywords, and use themes for image recolouring.

Recent work by Lin et al.\([80]\) has investigated color theme extraction from photographs. Lin
et al. [81] also investigate graphic design colouring, using a factor graph model trained on example designs. This work uses the color compatibility model of Chapter 3, published in O’Donovan et al. [109]. Yu et al. [169] also used this model for synthesizing outfits with compatible colors.

### 2.3 Image Aesthetics

Recently, large online datasets of rated photographs have been used to investigate image aesthetics. These datasets are generally used to train simple classifiers or regression models, with the learned weights analyzed to determine important components of the aesthetic rating.

Ke et al. [61] design features from photographic principles, including spatial distribution of edges, color distribution, blur, and low-level features like contrast and brightness. 60,000 rated photographs were used, and classification performed on the top and bottom 10%. Image blur, modeled as the inverse of the highest frequency present in the image, was found to be the most important feature for ratings. Fig. 2.2 shows examples of this model. Datta et al. [22] present a similar approach for classification and regression of ratings. They find the most important features include overall brightness and saturation, region saturation and size, and low depth of field. Luo and Tang [86] extract the subject of a photograph by finding a single non-blurred region, as use this to estimate features for classification. They find that image clarity (ratio of foreground/background blur) to be the most discriminative feature for prediction.

Researchers have recently turned to specialized feature sets. Obrador et al. [108] use only image composition features, including the common Rule of Thirds heuristic, i.e., that important
elements should be placed on the “Third lines” of an image. Nishiyama et al. [106] use only color harmony features, specifically a “bag-of-colour-patterns” approach, similar to the popular bag-of-words model. Dhar et al. [23] supplement the low-level features of Ke et al. [61] with high-level recognition features (e.g., face and animal detectors), as well as composition features like the Rule of Thirds.

In Chapter 3, we model the aesthetics of color themes in a similar manner as the above approaches. We also develop a large feature set, train a regression model from a large online dataset, and then perform simple feature analysis to investigate aesthetic rules.

In contrast to carefully designed features, Marchesotti et al. [90] use generic features from the image classification literature, including SIFT and color features. An advantage of the approach is that correlations between aesthetics and content are exploited, as the features are designed to capture image content. For example, the model may give higher ratings to portraits than to city scenes. Unfortunately, interpretability is lost in this approach. Related work by Karayez et al. [60] investigates different feature sets for aesthetic style classification, e.g., whether an image is ‘vintage’ or not. They compare various features, including GIST, color features, and features learned from deep neural networks, and find that neural network features perform best.

The previous work models aesthetics as a scalar value or binary label (“good” or “bad”). However, training on the mean score ignores the significant variance in the ratings. By contrast, Wu et al. [167] model the distribution of ratings using a structured SVM. They also improve results with a reliability function which weights training data based on the number of ratings.

### 2.4 Individual Aesthetic Preferences

As we described in the previous section, large datasets of visual objects are increasingly used to train aesthetic models. In these works, ratings are commonly averaged over all users to compute an overall prediction of aesthetic quality. However, it is well known that individual differences exist in aesthetic preferences [94], and some researchers have begun modelling differences between demographic groups and individuals.

Reinecke et al. [129] investigate aesthetic ratings for webpage design. They develop predictive models for perceived complexity and colourfulness of webpages, and find that these features are useful for explaining aesthetic ratings. In follow-up work, Reinecke and Gajos [128] examine website aesthetics for various demographic groups including gender, age, education, and nationality. They found significant variation between these groups, and model user preferences based on these demographic features. In Chapter 4, we also perform user modelling of aesthetics, but
instead of only using demographic features, we learn latent features for each user, permitting personalized predictions.

Researchers have also investigated individual preferences for image enhancement parameters. Kang et al. [59] explore personalized image enhancement, specifically color correction and contrast adjustment. They use a nearest neighbours approach, and copy parameters from the nearest user-modified image in a training set. However, while users prefer personalized photos to the originals, there is little improvement over automatic techniques like Picassa. Caicedo et al. [13] extend this work with collaborative filtering techniques. They first collect image adjustments from users, and define a probabilistic model which assigns each user to one of $k$ clusters. Each user’s observed enhancement vector is then modelled as a noisy version of a latent enhancement vector from a particular user cluster. Bychkovsky et al. [12] learn tonal adjustments for photographs. A dataset of 5000 images was collected, with tonal adjustments from 5 photography students on each image. The brightness remapping curve is projected to the first principle component analysis (PCA) dimension, and learned with Gaussian Process regression. Our work in Chapter 4 differs since we learn aesthetic preferences, not image enhancement parameters.

2.5 Design Layout Aesthetics

As mentioned earlier, graphic design covers a wide range of imagery. In this thesis, we focus on layouts for single-page graphic designs, such as posters and advertisements, since aesthetics plays a large role and there is relatively little prior work. However, there has been substantial work in automating layouts for primarily text-based documents such as articles [63]. In this case, templates and dynamic programming can be used to efficiently generate layouts [54, 20, 51] (Fig. 2.3). However, single-page graphic designs are more free-form and do not easily conform to templates or a linear read order. Harrington et al. [43] present an energy function to measure the aesthetics of a layout that also includes terms like alignment and balance; however, they do not show any designs and offer no evaluation. Balinsky et al. [7] describe measures of alignment in documents, as well as an aesthetics-driven layout engine for documents without a sequential read order [6]; however, only a single result is shown, and no evaluation is presented. Jahanian et al. [56] analyze the visual saliency of photographs to guide placement of text in magazine covers. Gonzalez-Morcillo [36] present a system for creating single-page graphic designs. However, this approach cannot learn different styles and uses a simple layout technique that normally aligns elements to margins.

The general approach of defining and optimizing an energy function is common for many
2.5. Design Layout Aesthetics

Figure 2.3: Adaptive templates from Jacobs et al. [54] (left), and probabilistic templates from Damera-Venkata et al. [21] (right).

layout problems, such as generic text and figure blocks [125], photo albums [33], route maps [2], and furniture layout [99, 168]. These approaches generally involve simple hand-tuned energy functions with a few terms such as alignment or balance. See Fig. 2.4 for examples.

A few approaches in design synthesis learn model parameters. In interface design, Gajos and Weld [31] define a model to specify the position and types of widgets. Users select between different interfaces and a margin-based learning approach sets the linear weights of the objective function. Vollick et al. [161] model layouts of labels of parts in technical diagrams. An energy-based model evaluates label layouts, and Nonlinear Inverse Optimization (NIO) learns parameters to create layouts in different styles. In Chapter 5, we use NIO in a similar manner to learn parameters for a graphic design layout model.

Analyzing document structure is another well studied problem, particularly for understanding digitally scanned and processed documents, including articles, book pages, or reports [89]. One basic problem is inferring the parts of the design and the overall structure. Another important task is analyzing the logical structure of design, for example, determining the title, abstract, and paragraphs in a document. A common approach for such analysis is grammar-based parsing, where parameters are either hand-tuned or learned using labelled training documents [141]. Talton et al. [149] present an approach that learns grammar production rules to parse web pages.

A related analysis problem is design segmentation. Rosenholtz et al. [132] segment user interfaces and infographics using orientation or lightness. Designers often use grids, either explicitly or implicitly, to organize elements [101]. Baluja [8] uses a grid-based segmentation of web pages for mobile browsing, while Krishnamoorthy et al. [68] hierarchically segment journal pages into rectangular regions.
Figure 2.4: Examples of various design layout problems including (a) photo albums [33], (b) route maps [2], (c) interfaces [31], (d) diagram labels [161], (e) furniture arrangements [99].
A related problem to our graphic design retargeting application is the retargeting of web pages to different display sizes. Kumar et al. [71] present a learning-based system for example-based web page retargeting. Mappings between DOM (Document Object Model) elements of two web pages are learned from user data, allowing style transfer and retargeting. Baluja [8] retargets by segmenting a web page into a 3x3 grid and magnifying these regions. However, these approaches are specific to web page design; single-page graphic designs are more free-form and graphical, and often do not have innate structure that maps easily to a DOM model. There has also been significant recent progress in image retargeting [133]. Liu et al. [83] use retargeting to optimize image composition using a simple objective function based on photographic principles. However, retargeting algorithms are inappropriate for design retargeting, which can modify element positions and scales. Brooks [11] employs design retargeting to synthesize movie posters which match the layout and composition of exemplar designs.

### 2.6 Design Interfaces

Design interfaces fall under the broad umbrella of creativity support tools. In his survey, Shneiderman [142] provides a set of design principles for these tools, including supporting exploratory search, enabling collaboration, and history-keeping of alternatives and changes. Tools should also have low thresholds and high ceilings, i.e., easy for novices to learn yet provide functionality for advanced users. In this work, our layout and font interfaces are designed with these principles in mind: enabling exploration, saving alternatives, and simple interactions for novices, while still providing useful functionality for more advanced users.

Exploring alternatives is a vital part of the design process. Gross and Do [41] present a prototyping interface that allows users to sketch drawings and store alternatives. The system also detects and parses configurations of layouts, such as furniture or circuit layouts, as well as inferring constraints between elements. Terry et al. [154] present Parallel Paths, an interaction technique where users can save and embed alternatives during the design process, and easily manipulate alternatives at a later point. The Juxtapose system [44] allows users to create alternatives using a source editor, as well as executing and evaluating the alternatives in parallel. Marks et al. [92] use a grid of alternatives to allow exploration of a high-dimensional parameter space. The d.tour system [131] enables exploration of web designs with a variety of search techniques including similarity, text, color, and attributes.

Prior work has also shown that providing alternatives produces better designs. Dow et al. [24] find that forcing users to create multiple graphic design alternatives, instead of refining
a single design, leads to improved designs. Lee et al. [76] present a web design interface which allows users to browse a corpus of related designs with simple attributes like background colour. They find that users who had access to these alternatives produced higher-quality designs.

There are also many commercial design interfaces, ranging in complexity from simple web-based template sites like Canva or Haikudeck, to complex systems like Adobe Illustrator. We incorporate techniques from these systems in our layout interface. Smart guide lines, such as in PowerPoint, are used to detect alignment between elements and snap them together.

Constraint-based interfaces are also common, such as Apple's Auto Layout or Dreamweaver's Fluid Grid Layouts. In these systems, constraints are defined by the user, for example, a minimum page margin or equal sizes for different elements, and a constraint solver computes an optimal layout. The Cassowary solver [3], used in Apple's Auto Layout, has also been used in constraint-based cascading style sheets (CCSS) [4], and constraint-based scalable vector graphics [5]. Constraint-based optimization is a powerful approach, but adding constraints is time-consuming and difficult, particularly for novices, and linear constraints do not allow the large-scale layout variations we demonstrate in Chapter 5 and 6.

Unfortunately, most of the automatic layout approaches described in the previous section are too time-consuming for an interactive design tool. One important exception is the work of Merrel et al. [99] on furniture layout. They present an interactive, suggestion-based interface which efficiently samples layouts on the GPU using parallel tempering [147], and shows refinement suggestions based on the user’s current layout. Our work is Chapter 6 is inspired by this approach. However, we tackle the more complicated domain of single-page graphic designs, show style suggestions, and investigate an adaptive interface where elements are automatically shifted.
3

Color Compatibility from Large Datasets

*It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.*

– Arthur Conan Doyle, *Sherlock Holmes*

Choosing colors is a difficult but crucial task for both amateur and professional designers. Designers often look for inspiration from many sources, such as art, photography, and color palette books. Color choice is guided largely by intuition and qualitative rules, such as theories of complementary colors and warm versus cool colors. It is generally believed that certain color combinations are harmonious and pleasing, while others are not. In the past two centuries, many conflicting theories of color compatibility have been proposed to explain these phenomena, but there has been little large-scale testing to validate or disprove them.

Online communities provide new ways for graphic designers to create and share color designs. Two websites, Adobe Kuler and COLOURLovers, allow users to create color themes, i.e., ordered combinations of 1-5 colors, though the vast majority have 5 colors. Each theme has a name, but is otherwise free of context. Users may rate, comment on, and modify previously-created themes. Over two million themes have been created on these sites, by tens of thousands of users. The datasets produced by these websites provide an opportunity for quantitative study of color theories and development of new color compatibility models.

This chapter employs online datasets to study color compatibility, with three main goals.

---

A version of this chapter was published in ACM Transaction of Graphics (Proc SIGGRAPH), 2011. [109]. This paper was a collaboration between the author, Aseem Agarwala, and Aaron Hertzmann. The project page is at www.dgp.toronto.edu/~donovan/cfcolor, and contains code and datasets used in this chapter.
First, we test new and existing theories of color compatibility. For example, we test to what extent certain hues or hue templates may be preferred by viewers. Second, we learn quantitative models to rate the quality of a color theme. Third, we demonstrate simple prototypes that apply these learned models to tasks in color design, including improving existing themes and extracting themes from images. Together, these prototypes illustrate how the development of effective color compatibility models could be useful for various tasks in graphic design and computer graphics.

Our studies are based on three datasets, each of which comprises a collection of color themes and their ratings. We derived two datasets from Kuler and COLOURLOvers, and created the third using Amazon Mechanical Turk (“MTurk”). These datasets exhibit different advantages and disadvantages. For example, Kuler users have more exposure to color theory than MTurk workers, while MTurk data is collected in a more controlled fashion. However, taste in color can vary widely, and users in these datasets have varying goals, backgrounds, and viewing environments; not surprisingly, there is substantial variation. Nonetheless, analysis of the data reveals many regularities and patterns.

We first analyze these datasets to understand which colors people use, and how colors are combined. Our main observations are as follows. User-created themes are far from random; there are areas of higher density in the space of 5 colors, and themes farther from these areas tend to be rated worse. People also have strong preferences for particular colors. The data reveals a preference for warm hues and cyans in color themes, which is distinct from preferences for purples and blues with single colors. Hue templates, the most popular models of color compatibility, are tested in several ways, and no evidence is found that they predict compatible colors. We examine the number of distinct hues people prefer in a theme, and find users generally prefer themes which are neither too simple (i.e., monochromatic), nor too complex (more than 2-3 different hues). Further MTurk experiments indicate that theme names usually do not affect the rating, though evocative names can have an impact.

We offer a new color compatibility model for predicting ratings, and examine which features of color themes are most important. Our model is distinct from previous work in that it uses a large number of features in many color spaces. The model is learned by linear regression with an L1-norm, thereby selecting the most relevant features for predicting the aesthetic rating. In particular, lightness features are important; dark themes are poorly rated and gradients from light-to-dark or vice-versa are preferred. Choosing popular adjacent color pairs is important, and theme colors should not be too similar to each other.

Aside from their scientific value, effective compatibility models would be useful for numer-
3.1 Datasets

Our work employs three datasets which we first describe at a high level. See Appendix A for detailed statistics of the datasets and how they were created. The Kuler dataset comprises 104,426 5 color themes created by visitors to the Kuler website (kuler.adobe.com). Color themes are also ordered, with the adjacency specified by the user. Each theme can be rated on a discrete scale of 1 to 5 (Figure 3.1). The dataset includes 327,381 ratings from 22,376 users. Except where noted, in all of our experiments we omit themes with fewer than 2 ratings, leaving 46,137 themes with 266,239 ratings.

The COLOURLovers site (www.colourlovers.com) includes over three million 2-5 color themes created by users. While themes are not rated directly, users may “heart” any theme. The total number of hearts $h$ and user views $v$ is given for each theme. We define a rating function $r(h, v)$ that computes a score based on $h$ and $v$ (see Fig. 3.2 and Appendix A). Our COLOURLovers dataset includes 383,938 five color themes downloaded from the COLOURLovers
Figure 3.2: Views vs. hearts for COLOURLovers dataset. The red line is the fit of the histogram means $\bar{h}(v)$, the green is the fit of the standard deviations $\sigma(v)$. Ratings are estimated as $r(h, v) = (h - \bar{h}(v))/\sigma(v) + 3$. For example, at 800 views, a theme requires more than 10 hearts to receive a rating over 3.

Websites. Ratings for 178,086 themes with over 100 views were estimated.

In order to obtain data under more controlled conditions, we created a third dataset using Amazon Mechanical Turk (MTurk dataset). We first selected 10,743 Kuler themes covering a range of poorly-rated to highly-rated themes; only themes with at least three user ratings were considered. MTurk tasks required each participant to rate 30 themes on a discrete scale from 1 to 5, with duplicate themes added to check for consistency. Each theme was rated by 40 participants, with 1,301 participants total.

These datasets each have advantages and limitations. Kuler and COLOURLovers are used by highly motivated designers, including professionals with an interest and aptitude for colour. However, no demographic information is available to control for the different tastes, backgrounds, or goals of users. Hence, these datasets may mix together very different aesthetics and design goals. Kuler and COLOURLovers have interface biases, namely, specific affordances for creating themes with one of several standard hue templates. Both sites encourage users to name their themes.

The data also includes numerous rating biases. Themes may be rated non-uniformly—with some exceptions, most themes in Kuler have very few ratings, and our ratings for COLOURLovers are inferred (see above). Both sites promote highly-rated themes, so popular themes are more likely to get more ratings, the so-called “rich-get-richer” effect [26]. Also, a user’s opinion of
a theme can affect whether or not they rate it. For example, a few users only rate themes with 4 or 5 star ratings. When missing ratings are not “missing-at-random,” learned estimators can perform poorly on random test data [93].

By contrast, MTurk users are assigned random themes to rate. MTurk also allows us to ensure that we get sufficient numbers of ratings for a wide range of themes; we can avoid community biases and naming biases. MTurk workers are much less likely to be professionals with interest or experience in working with colour. MTurk experiments are less controlled than the in-person experiments common in the psychological literature, but they can be run on a far larger scale. Heer and Bostock [45] have demonstrated the viability of MTurk for graphical perception experiments by comparing to classic results from the literature.

All our datasets may have variation due to differences in users’ monitors, viewing conditions, and color blindness. Color calibration in online experiments is a challenging problem. However, most graphics applications are aimed at uncalibrated viewing, so finding colors which are compatible on average over many viewing conditions is important for graphic designers. Lastly, our conclusions are restricted to the color gamut of conventional monitors.

3.2 Model-Free Data Analysis

In this section, we consider general questions of color compatibility, independent of specific learning algorithms. First, the data density of user-created themes is measured to evaluate whether themes are uniformly scattered in the space of 5 colors, or whether there are areas of higher density. Preferences for individual colors and colors in combination are examined, as well as the popularity of adjacent hue pairs. Hue templates are evaluated in detail, specifically looking at the rotational invariance assumption, as well as the prevalence of templates not implemented in the user interfaces. Based on our experiments with hue templates, we define hue entropy, a theme complexity metric which roughly corresponds to how many distinct hues are present in a theme. Lastly, the impact of theme names on user ratings is evaluated.

3.2.1 Distribution of Themes

To what extent are user-created themes spread uniformly distributed or densely clustered in the space of possible themes? We can explore this spread by measuring the distance from a single theme to a separate set of user-created themes. Fig. 3.3 (left) shows a histogram of distances from 10,000 Kuler themes to a disjoint set of 27,000 Kuler themes. For comparison, the distribution
of distances from 10,000 themes uniformly sampled in each of RGB and HSV space is shown as well. A similar histogram for COLOURLovers to Kuler themes is also shown. The distance was calculated by the sum of CIELab distances for each color in the theme. The mean distance to the nearest 10 themes indicates the distance to nearby themes, and the spread of themes throughout the space. The shape of these plots indicate the degree of clustering: Kuler and COLOURLovers demonstrate similar clustering, whereas the random themes are considerably more spread-out. Kuler and COLOURLovers themes are therefore not spread uniformly in the space of 5 colors: there are definite regions of higher density.

Given the non-uniform distribution of themes, does proximity to other themes help indicate the rating of a new theme? Fig. 3.3 (right) plots mean rating for a test theme against the distance to the 10 Nearest Neighbors. A downward trend appears in all three datasets (especially in Kuler and MTurk), indicating that unusual themes are more likely to have lower scores. Hence, a new theme that is similar to existing themes is more likely to receive a higher rating.

**Figure 3.3: Data density vs. rating. Left:** Theme distance to 10 Nearest Neighbors (NN) in a Kuler training set. This figure suggests that the Kuler and COLOURLovers datasets are similar, and quite different from data randomly sampled in RGB or HSV space. **Right:** Distance to Kuler training set vs. mean rating. Themes similar to the other data are more likely be highly rated. Error bars show 2 std error. These results suggests there are dense regions of the color theme space with higher aesthetic scores.
3.2. Model-Free Data Analysis

3.2.2 Preferred Colours and Color Pairs

We next examine overall preferences for colors, both for single colors and in combination. Previous work on color preferences, based on in-person surveys of small groups of individuals (see included references in Ou et al.

Fig. 3.4 indicates the hue histogram for the Kuler and COLOURLovers datasets, and indicate greater density for warm hues (red, orange, yellow) and blues. A large spike appears at pure red, perhaps because red lies at the ends of the hue sliders in both interfaces. Fig. 3.5 shows the average rating assigned to themes containing that hue.

However, these ratings mix together the contribution of each color to the rating of the theme. We therefore consider an approach to “unmixing” the effect of color preferences on theme ratings. We discretize hues, and treat each distinct hue \( j \) as having a hidden “quality” \( q_j \). Suppose a theme \( t \) has rating \( r \). We model this theme’s rating as arising from the average of the qualities of the \( N \leq 5 \) colors of a theme as:

\[
r = \frac{\sum_{j \in t} q_j}{N}
\]  

\[ (3.1) \]
3. Color Compatibility from Large Datasets

Figure 3.5: Color preferences. **Left:** Mean rating of themes containing each hue, and individual color ratings from MTurk. While the 3 datasets vary, they display similarities in relative hue preference, particularly compared with individual color ratings. **Right:** Unmixed rating quality for hues.

The Kuler data provides us with a large collection of pairs of themes and rankings. Each theme has a rating and set of colors, yielding a linear equation of the form of Eqn. 3.1. We can directly estimate the qualities $q$ of each color by solving the resulting system of equations in a least-squares sense. Only saturated and light colors are considered ($c_{\text{sat}} > \tau_{\text{sat}}$ and $c_{\text{val}} > \tau_{\text{val}}$), and themes with no saturated or light colors are ignored. In Fig. 3.5, we plot the results for the average ratings of all themes containing each color, along with the unmixed weights. Note that while the results are noisier, particularly for MTurk due to the fewer constraints, the same relative preference for hues is apparent with more exaggerated peaks and valleys.

We find that single-color preferences do not match preferences for color combinations, a result that echoes previous findings from in-person studies [119]. The preferences of Kuler and COLOURLovers users appear more similar to each other than to the MTurk users, which might reflect that the former are more likely to be professional designers. We conjecture that designers may be rating themes on their usefulness, whereas MTurk users are asked to rate solely based on visual appeal.

In Fig. 3.6 and 3.7 we plot the distribution of colors with respect to hue versus saturation, and hue versus value for both datasets. The distribution of colors from both datasets is very similar, showing a strong preference for bright warm colors and cyans. Note that fully saturated colors are extremely popular for all hues. However, de-saturated yellows are common, with reds tending to be more saturated. Greens are mostly lighter and unsaturated.
Figure 3.6: Kuler color density of hue versus saturation (left), hue versus value (right). Density is visualized in greyscale, with high-density regions as white and low-density regions as black. The grid-like pattern is a product of the color spaces used in the color picker interfaces. For example, the RGB value for pure green is (0,255,0). Varying the brightness of the color to black (0,0,0) is facilitated by sliders in both Kuler and COLOURlovers, and produces the visible vertical line in the right figure.

Figure 3.7: COLOURlovers color density of hue versus saturation (left), hue versus value (right). Density is visualized in greyscale, with high-density regions as white and low-density regions as black.

Fig. 3.8 (top) shows the pairwise co-occurrence of adjacent hues in themes. Fig. 3.8 (bottom) shows the joint probability over all hues in a theme. That is, the probability that two hues will be in the same theme, regardless of adjacency. The overall joint distribution of hues in a theme is quite similar to the pairwise distribution. Warm hues around yellow and red have strong adjacency, and yellow and cyan are often paired with many other hues. Green and purple are relatively unpopular hues, and are more commonly paired with similar hues.

3.2.3 Hue Templates

Perhaps the most prominent theory of color compatibility is the notion of hue templates: fixed sets of rotations around the color wheel which produce compatible colors (see Figure 2.1 for examples). Here we investigate whether these templates describe the themes that users create, and whether the use of a template predicts better ratings. Previous research for 2 and 3 color combinations did not find complementary and triadic hue templates to be harmonious [117, 148] but these studies were limited to 17 and 9 participants, respectively. To our knowledge, ours is the first study exploring compatibility of combinations of up to five colors, or with user-created
Figure 3.8: Top Row: Pairwise probability of adjacent hues. High brightness indicates higher probability. Top left, COLOURLovers dataset. Top middle, Kuler dataset. Top right, Kuler dataset remapped to the BYR color wheel used in the interface. Diagonal lines correspond to hue templates and show a lack of rotational invariance in the dark bands around purple and green. Lack of smoothness in the data is apparent in the BYR histogram, also visible in Fig. 3.4(top). Bottom Row: Joint probability over all hues in a theme. Bottom left, COLOURLovers dataset. Bottom middle, Kuler dataset. Bottom right, Kuler dataset with hues remapped to BYR color wheel used in Kuler interface. The joint probability of hues corresponds closely with the pairwise probability.

The use of color templates can be clearly seen in the Kuler pairwise color histogram (Fig. 3.8): diagonal lines in the histogram correspond to fixed rotations about the color wheel. These lines can be explained as a result of interface bias: both Kuler and COLOURLovers provide tools for creating themes with templates, but they are harder to find and use in COLOURLovers. This figure suggests that designers do not create themes that match templates unless encouraged to do so by the interface. These histograms also strongly suggest that, contrary to belief, preferences are not rotationally invariant about the color wheel: green’s complement is purple, yet these plots
suggest users prefer to pair green with blue or yellow instead. On the other hand, orange often pairs with cyan, its complement on the hue wheel.

We investigate hue templates in more detail by assigning each theme to the closest template; see Appendix B for details of the theme-template distance. In Figure 3.9 we show all the hue templates for COLOURLovers, Kuler, and Matsuda. Figure 3.9 gives a histogram of distances from themes to the templates implemented in Kuler and COLOURLovers, as well as to Matsuda’s templates. A commonly-used template should appear as a spike at zero distance. We see that themes implemented in the Kuler interface do appear often, whereas none of Matsuda’s additional themes do. This indicates that designers are not gravitating to Matsuda’s templates of their own accord. The histogram also shows that monochromatic (i), analogous (V), and complementary (I) templates are the most popular. These are the most elementary and basic templates. COLOURLovers shows even less use of templates, as can be seen in Figure 3.8. In the COLOURLovers interface, templates are harder to find and utilize than in Kuler. These results show that people only gravitate towards the most basic templates like i, V, and I, which are implemented in both interfaces (Fig. 3.9).

Does the distance to a hue template help predict ratings? Fig. 3.10 plots the rating of themes as a function of distance to their nearest template. Distance to the template does not appear to be closely related to the rating. However, in Kuler and COLOURLovers, themes which are very close to templates have slightly lower scores. We hypothesize these templates are created by more inexperienced users, or the community penalizes themes close to the interface defaults. More importantly, the MTurk results have no interface biases, and also show little evidence that template distance affects rating.

Finally, in Figure 3.10 we also assign themes to their nearest template and plot the histogram count along with mean ratings with standard deviation and 2 standard error. The results show a great deal of variation, but generally, themes distant from a template do not score lower than themes nearer a template. Certain templates are more popular than others, particularly simpler templates like V and i, which both indicate a set of nearby hues. The R and X templates which have 3 and 4 hues spread equally across the hue wheel are among the least popular, as are greyscale themes (template N).

In short, although users appear to use templates built into the Kuler interface, we find little evidence that people gravitate to templates naturally, or that matching a template produces higher scores.
Figure 3.9: Top Row: template distance in Kuler dataset for interface-implemented templates, and for the rest of Matsuda's templates. Bottom Row: template distance for COLOURLovers dataset for interface-implemented templates, and for the rest of Matsuda's templates. Note the spike around zero for templates implemented in the Kuler interface which is mostly lacking in the COLOURLovers data. The spike at 60 degrees for the V template is caused by monochromatic themes with a single accent color between 30-60 degrees. These results strongly suggest that users do not naturally gravitate towards fixed hue templates.

3.2.4 Hue Entropy

As noted in the previous section, simple themes—typically consisting of one hue, two hues, or a blend—tend to be more popular than complex ones. Here we propose hue entropy $H(t)$ as a
3.2. Model-Free Data Analysis

Figure 3.10: Top left: Nearest template distance vs. theme rating. Themes that closely match templates tend to score lower than those that do not. Beyond this, increasing the distance to a template does not significantly affect the rating. See Appendix B for description of distance metric. Top right: Template histogram. Themes assigned to a template if distance < 90 degrees. Bottom: Template mean ratings. For each set of themes assigned to a template, we calculate the mean ratings with standard deviation and 2 standard errors. These results show that people only gravitate towards the most basic templates like i, V, and I, which are also implemented in the interfaces, and many themes do not correspond to a hue template. There is a great deal of variation, but generally, themes which do not match a template do not score lower than themes nearer a template. However, themes with a greater spread of hues, such as those near the R and X templates, do have lower ratings.

A single measure of the simplicity of a theme \(t\). Let \(\theta_1, \ldots, \theta_5\) be the hues in a theme, represented as angles. We convert these hues into a probability distribution as a mixture of von Mises distributions: 

\[
p(\theta) \propto \sum_{i=1}^{5} \exp (\kappa \cos(\theta - \theta_i))
\]

The hue entropy is then the entropy of this distribution,
Figure 3.11: **Hue entropy vs. theme ratings.** The entropy values for equally-spaced hues are, from 1 hue to 5 hues: (4.62, 5.29, 5.65, 5.81, 5.87) where the first 4 correspond to the i, I, R, X templates. The analogous template V has an entropy value of 4.89, the C template is 5.33. Monochromatic or complex themes (i.e., more than 2-3 hues) tend to rate worse. Themes too close to an interface template result in a significant ratings drop in the Kuler dataset (see Sec. 3.2.4).

computed numerically. \( \kappa = 2\pi \) was selected to maximize the regression score for a set of training themes. Hue entropy is lowest when all values of \( \theta \) are identical \( (H = 4.62) \), and highest when they are uniformly spread about the circle \( (H = 5.87) \). Colours spread over a narrower range will have lower entropy than a wide range (e.g., red-to-magenta has lower entropy than red-to-green).

Figure 3.11 shows the relation of hue entropy to theme rating in the three datasets. The data shows a clear trend: for the Kuler and COLOURLovers ratings, there is a preference for entropies roughly in the range 4.7–5.4. This corresponds to themes with about 2-3 hues. MTurk ratings are more uniform, with no penalty for monochromatic themes, though scores trend downward for themes with more colors. This highlights a difference between the datasets: Kuler and COLOURLovers users are likely evaluating the usefulness of themes for design tasks, and monochromatic themes are rarely useful for design. By contrast, MTurk users are rating purely based on visual appeal. The figure also shows clearly that using templates provided by the interface correlates with lower ratings.

### 3.2.5 Theme Names

All themes on Kuler and COLOURLovers are given a name. Names provide some context to themes, describing what they are meant to evoke or how they might be used. Palmer and Schloss
3.3. Learning Color Compatibility

Figure 3.12: Effect of theme names. For each theme, the name, and the average MTurk rating without and with name, respectively, are shown. In most cases (95% of our test themes), names have no statistically significant impact on rating. However, in a few cases, names improve scores (top row), or worsen them (bottom row).

[121] provide evidence that color preferences are affected by real-world associations, and color names might affect these associations. To what extent does the name affect the rating of a theme?

Using MTurk, we asked 40 participants to rate 216 themes where each participant was shown all themes either with or without their original names from Kuler. Of the 216 themes, only 10 showed a significant difference (using the Mann-Whitney U-test) in scores with the name versus without. Hence, names affect rating only in a minority of cases, though as Fig. 3.12 illustrates, evocative names can have a significant impact. The mean of the absolute difference for named vs. unnamed ratings was 0.20, median 0.16, and max 0.88 (the name “ml126” decreased the rating from 3.56 to 2.68).

3.3 Learning Color Compatibility

We now describe methods for learning to rate color themes. These learned models allow us to perform additional analysis of the data, and to create new colour-selection applications.

The input data comprises pairs \((t_i, r_i)\), where \(t_i \in R^{15}\) represents theme \(i\), and \(r_i \in [1...5]\) is the mean user rating for this theme. Our goal is to predict the mean rating \(r_{new}\) for a new theme \(t_{new}\). For performance, we only use themes with at least two user ratings each. We learn
separate models for all datasets. For the COLOURLovers dataset, we restrict our regression tests to 60,000 randomly-selected themes.

**Feature vectors.** As input to the learning algorithm we define a *feature vector* \( \mathbf{y} \) that can be computed from any input theme \( \mathbf{t} \). However, due to the exploratory nature of this work, we do not know in advance what the best features are. The feature vector comprises a large set that might be useful; finding relevant features is valuable for understanding color compatibility in general.

The feature vector has 334 dimensions and is constructed from four color spaces: RGB, CIELab, HSV, and CHSV\(^1\). We create the following features in each space: the five colors themselves, colors sorted by lightness, differences between adjacent colors, sorted color differences, mean, standard deviation, median, max, min, and max minus min across a single channel. Differences in hue are computed with wraparound. Note that some features are redundant. For example, lightness/value is represented in many spaces. Since many themes lie along lines or planes in color space, we also include plane-fitting features in RGB, CIELab, and CHSV. A 2D plane is fit to the 3D color coordinates using principle component analysis (PCA) \([123]\) and the plane normal, eigenvalues, and sum-of-squared error used. Hue entropy is also included (Section 3.2.4). All features are normalized to the range \([0...1]\).

Lastly, we use the color histograms from the Kuler training set to produce scores for individual colors and pairs of colors. Let \( p_c \) be the percentage of colors in the training themes with hue \( c \) (Figure 3.4), \( p_{bc}^a \) be the percentage of adjacent colors \( b \) and \( c \) (Figure 3.8), and \( p_{bc}^l \) be the percentage of colors \( b \) and \( c \) in the same theme. A list of hues from saturated colors and light colors (both are determined by thresholding) is first created from a theme. The probability of each hue is taken, and the mean, standard deviation, min, and max computed. Features are also computed with log probabilities. The same features are then computed for the pairwise probabilities \( p_{bc}^a \) and \( p_{bc}^l \). When there are no saturated or light colors, the features are set to 0.

**Regression.** In regression, we learn a continuous mapping from feature vector \( \mathbf{y} \) to rating \( r \). We test the following regression algorithms: LASSO \([155]\), robust linear regression using Iteratively-Reweighted Least-Squares, Support Vector Machine regression (RBF kernel), and \( K \)-Nearest Neighbors.

Given an input feature vector \( \mathbf{y} \), the \( K \)-NN regressor computes the \( k \) training feature vectors \( \mathbf{y}_j \) most similar to \( \mathbf{y} \), and returns a weighted linear combination of their mean ratings \( r_j \), taking

\(^1\)A space where hue \( \theta \) and saturation \( s \) are remapped to Cartesian coordinates: \( d_1 = s \cos(\theta) \) and \( d_2 = s \sin(\theta) \).
3.3. Learning Color Compatibility

Figure 3.13: Example themes, including well-rated (top row), poorly-rated (middle row), and themes with high regression error (bottom row). User rating \( r_i \) and LASSO regression \( r(t_i) \) results shown for each theme, with LASSO from MTurk training set. All themes and scores are from the MTurk testing set.

into account the distances between feature vectors, the number of ratings \( N_j \), and the rating variance \( v_j \):

\[
\begin{align*}
    r_{knn}(y) &= \sum_j w_j(t) r_j / \sum_j w_j(t) \\
    w_j(y) &= \frac{\exp(-||y-y_j||^{\sigma_{off}})}{1+\exp(-N_j^{\sigma_{cnt}})}^{\sigma_{var}}
\end{align*}
\]  

(3.2) (3.3)

where \( K=50, \sigma_{off}=1, \sigma_{cnt}=0.05, \sigma_{var}=0.5 \), set by cross-validation.

As a baseline, we use a fixed regressor that outputs the mean rating of all themes \( \bar{r} \). The SVM RBF kernel width was \( \gamma = 4 \), and the margin \( C = 0.5 \). To reduce overfitting for SVM and KNN, we apply them only to the top \( N \) features selected by the LASSO regressor. All parameters were set with cross-validation, which selected \( N = 40 \).

We find that the LASSO algorithm generally gave best results, and also performs automatic feature selection. Sample results are shown in Figure 3.13, and Table 5.1 shows the regression results using a 0.6/0.4 train/test split. The LASSO regressor is a linear function of the features:

\[
r(t) = w^T y(t) + b,
\]

learned with \( L_1 \) regularization:

\[
    \arg \min_{w,b} \sum_i (w^T y_i + b - r_i)^2 + \lambda \|w\|_1
\]

(3.4)
Table 3.1: **Regression and classification results**, including mean average and squared error (MAE/MSE). For classification, the LASSO regression output is compared to the mean theme rating to distinguish “good” themes from “bad.”

<table>
<thead>
<tr>
<th>Set</th>
<th>Fixed</th>
<th>LSO</th>
<th>IRLS</th>
<th>SVM</th>
<th>KNN</th>
<th>Class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL MAE</td>
<td>0.572</td>
<td>0.521</td>
<td>0.523</td>
<td>0.531</td>
<td>0.533</td>
<td>64.2%</td>
</tr>
<tr>
<td>CL MAE</td>
<td>0.703</td>
<td>0.664</td>
<td>0.654</td>
<td>0.650</td>
<td>0.674</td>
<td>60.1%</td>
</tr>
<tr>
<td>MT MAE</td>
<td>0.267</td>
<td>0.179</td>
<td>0.179</td>
<td>0.182</td>
<td>0.205</td>
<td>77.4%</td>
</tr>
<tr>
<td>KL MSE</td>
<td>0.525</td>
<td>0.448</td>
<td>0.449</td>
<td>0.466</td>
<td>0.470</td>
<td>64.2%</td>
</tr>
<tr>
<td>CL MSE</td>
<td>0.763</td>
<td>0.688</td>
<td>0.695</td>
<td>0.725</td>
<td>0.708</td>
<td>60.1%</td>
</tr>
<tr>
<td>MT MSE</td>
<td>0.115</td>
<td>0.052</td>
<td>0.053</td>
<td>0.053</td>
<td>0.068</td>
<td>77.4%</td>
</tr>
</tbody>
</table>

**Figure 3.14:** **Regression results for MTurk testing set.** Predicted ratings compared to human ratings for all themes.

The optimal parameters $w$ and $b$ are computed via a convex optimization [29] (glmnet package), with $\lambda = 0.00016$ selected by cross-validation. We use the LASSO model for all further regression tests. On the MTurk dataset, LASSO gives a 32% decrease in mean absolute error (MAE) over the baseline, and a 53% decrease in mean squared error (MSE). Table 5.1 shows several examples and we plot the full test set in Figure 3.14.

In Figure 3.15 we plot the effect of increasing the minimum number of ratings for each theme. A minimum number of two ratings was chosen as this provided a large gain over the baseline estimator while still preserving a large number of themes.
3.4. Model-Based Data Analysis

We now perform further analysis and experiments on the datasets using regression. Specifically, we investigate which features are most predictive of the rating, and test the value of learning different models for different subsets of users.

3.4.1 Important Features

Because LASSO uses a linear predictor, inspecting the feature weights \( \mathbf{w} \) of a learned model gives a sense of which features are most predictive of rating for that dataset. All features are normalized to the range \( 0...1 \), so weights are directly comparable. However, it is important to remember that the LASSO ratings depend nonlinearly on the input theme since the features are nonlinear. Therefore, examining individual weights gives only a partial picture of the predictor’s behavior.

**Classification.** We also learn classifiers to distinguish “good” themes from “bad”. Classification provides another way to test the predictability of the data; we do not use classification in any of our applications. A training point is marked as “good” if its mean rating is above the mean rating of all themes, and “bad” otherwise. Table 5.1 shows classification results using the LASSO regressor. We also experimented with other classifiers (Logistic Regression, Gentle AdaBoost, SVM) but found results only improved slightly (< 1%).

**Figure 3.15:** Left: effect of increasing the minimum number of ratings for Kuler dataset. Right: histogram of theme count for each test.
Lightness features are among the most predictive of good themes. MTurk's most important feature is Mean Lightness, indicating a preference for overall bright themes. However, in Kuler, the Max Lightness feature is important, indicating that users like darker colors, as long as there is at least one bright colour. The feature measuring difference between min and max lightness is highly weighted, further indicating that a spread of lightness is important. These features support previous research showing that lightness contrast is important for harmony [148, 136]. However, all models heavily penalize a high standard deviation in lightness, suggesting that pairing several bright and dark colors is a poor choice (e.g., Figure 3.13 (center)). Combined, these two features promote high contrast with low standard deviation, that is, a gradient; indeed, many highly-rated themes in MTurk are simple gradients from light to dark, e.g., Figure 3.13 (upper-right).

All three datasets have a positive weight on the mean of hue probability $p^i_{bc}$, indicating that pairwise relationships between colors are important, as are choosing warm hues and cyans (which have higher probability). A significant negative feature for all datasets is the min of the pairwise hue probabilities, $p^i_{bc}$; a large min value indicates that all the colors in a theme are the same or similar, since, as can be seen in Fig 3.8, $p_{bc}$ is largest when $b$ and $c$ are the same. Hence, a set of good colors should be reasonably popular, but not too similar.

Some important differences arise between models. The most positive feature in the Kuler model is the standard deviation in CIELab's b dimension (roughly, blue to yellow). This feature favors a bimodal set of similar colors instead of a transition of different colors. A high weight on the max-minus-min in this dimension keeps the colors from becoming too saturated. A similar preference for blues and yellows can be found in the COLOURLovers dataset in a CHSV standard deviation. The lack of this feature in MTurk may reflect some of the differences in color preference (Figure 3.5).

Hue entropy is an important feature for all models. For MTurk and COLOURLovers, it is negative, indicating a penalty for having too many distinct colors. In Kuler, entropy has a positive weight. Since the relationship between entropy and rating is roughly parabolic for Kuler, this suggests that the penalty for having too few colors (e.g., monochromatic), outweighs the penalty for having too many colors.

### 3.4.2 Ratings Across Datasets

In order to validate the COLOURLovers’ estimated ratings $r(v, h)$, in Fig. 3.16 we plot the predicted ratings using all three LASSO models on a single dataset (50,000 COLOURLovers themes
Figure 3.16: Evaluating COLOURLovers estimated ratings. Predicted ratings from each model are compared with a COLOURLovers test set. The estimated rating model matches the Kuler model closely, but not the MTurk model. This result suggests that our estimated ratings are reasonable, and that color preferences of Kuler and COLOURLovers users differ from those of MTurk users.

<table>
<thead>
<tr>
<th>Set</th>
<th>KL Select</th>
<th>CL Select</th>
<th>MT Select</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL MAE</td>
<td>0.552</td>
<td>0.546</td>
<td>0.545</td>
<td>0.572</td>
</tr>
<tr>
<td>CL MAE</td>
<td>0.692</td>
<td>0.686</td>
<td>0.686</td>
<td>0.703</td>
</tr>
<tr>
<td>MT MAE</td>
<td>0.207</td>
<td>0.217</td>
<td>0.201</td>
<td>0.268</td>
</tr>
</tbody>
</table>

Table 3.2: Regression MAE using only top-20 LASSO features selected from another dataset. The similar performance suggests significant overlap in the features relevant for each dataset.

not used for training). The strong positive trend between COLOURLovers and Kuler validates the estimated ratings. Between the COLOURLovers and MTurk ratings, a positive trend appears at lower scores, flattening out for higher scores. This indicates a higher discrepancy between COLOURLovers and MTurk users than between COLOURLovers and Kuler users. A similar comparison showed an intermediate trend between MTurk and Kuler.

To further explore the overlap between dataset features, we choose the top 20 features (according to absolute value) from each dataset and trained the other datasets on only those features. Table 3.2 shows that performance is similar for most models regardless of which dataset was used to select the features, indicating a significant overlap in features relevant for each dataset.
3.4.3  Mixture-of-Experts Collaborative Filtering

Color preferences often vary across individuals \[121\], suggesting that collaborative filtering might be used to improve individualized predictions. We test the presence of variations in style as follows. We first selected the 597 Kuler users that had rated at least 50 themes. We say that two users agree on a theme if both users rate that theme higher than their own respective mean ratings, or if both users gave ratings less than their means. The average agreement across all these users and themes is only 52\%, only a slight improvement over chance. However, it is possible to cluster users into groups and only compare their intra-cluster agreement. A simple clustering algorithm was used where each user was repeatedly compared against the agreement with all users in each cluster and re-assigned to the cluster with most agreement. For 10 clusters, within-cluster agreement jumps to 81\%.

We learned a Mixture-of-Experts (MoE) \[55\] model to predict color ratings by collaborative filtering. For each cluster, we first trained a LASSO regressor. Given a new user with some current ratings, the MoE first identifies which cluster’s regressor best fits the new user’s ratings. That regressor then predicts all missing scores for that one user. A set of test users with 25 to 50 ratings were chosen. Given 1 rating we choose the best cluster based on a score from a single sample theme. The resulting mean average error on the test set is 1.30. For 5 ratings, MAE=1.07, for 10 ratings MAE=1.03, for 20 ratings MAE=1.00. By contrast, if a single cluster is used for all expert users, the MAE is 1.30, indicating a 30\% improvement over using the same model for all users.

3.4.4  Clustering By Demographics

Another question is how different demographic groups compare. The demographics of the MTurk workers were 504 Indians, 351 Americans, and 146 from other countries. There were 593 female and 410 male participants. 548 participants were age 15-30, 300 were age 31-45, and 152 were age 46+. A LASSO model was first trained on each group. To account for improved scores for groups with more participants, each theme was given the same number of ratings from each group. Women’s scores were generally more varied than men, though prediction was similar (MAE: 0.26 vs 0.25). Indians rated more consistently than Americans and had lower prediction error (MAE: 0.54 vs 0.58). Prediction error also tended to decrease with age (MAE: 0.50/0.49/0.48). These results suggest that demographics groups do rate differently. However, a further test comparing models trained on demographic groups to a model trained on the full dataset revealed little improvement in prediction. This result suggests that personal preferences
3.4. Model-Based Data Analysis

Figure 3.17: Distance of an optimized color from the original compared to the theme rating. A downward trend indicates that the model generally suggests colors which are closer to the original for highly rated themes (where the original color choice was likely good) than for poorly-rated themes (where the original color choice was likely poor).

outweigh demographic preferences, so pooling between groups helps performance overall.

3.4.5 Color Suggestion Distance

How good are color suggestions made by our model? A simple test is to select a random color from a theme, set it to grey, and optimize for the best possible color using our model (see Section 3.5.1 for optimization details). Since the themes were human-rated, we have an estimate of the original colour’s quality. When the theme is poorly rated, we expect the original color was badly chosen, so our model will likely choose a more distant colour. However, when the theme is highly rated, we expect that the user has chosen a good colour. So we expect that on average, our choice would be closer. We can then plot the distance from original to optimized color (in CIELab) compared to the human rating. If the model suggests good colors on average, we expect to see a downward trend.

In Figure 3.17 we plot the results for themes from the Kuler and MTurk test datasets (4,861 and 4,291 themes respectively). We only use the MTurk and Kuler datasets as both have ground-truth human ratings. Both models have a downward trend which helps validate our model. For Kuler, the increased noise is likely since the low numbers of ratings per theme create more variance along the $x$-axis.
3.5 Applications

Choosing colors is a challenging problem in many design scenarios. We now demonstrate several simple prototype applications that illustrate how our model may be useful in design and computer graphics. In each case, we make use of a LASSO regressor that, given a color theme $t$, outputs a predicted rating $r(t)$. Except where noted, we use a regressor trained on the MTurk data, as we find this gives the best results when testing applications on MTurk users.

3.5.1 Theme Optimization

First, we use our learned model of color compatibility to improve existing themes. Given an input theme $t_{in}$, we seek a similar theme $t$ with the highest rating. The ratings given by our model depend on the order of the colors. Hence, the simplest way to improve the rating is to search for the best permutation of the colors: $t = \arg \max_{\mathcal{P}(t)} r(t)$, where $\mathcal{P}(t)$ denotes the 120 permutations of a theme $t$.

More generally, we can search for a theme that maximizes the score while staying within a given distance $d_{\text{min}}$ of the input theme:

$$\arg \max_t \max_{t' \in \mathcal{P}(t)} r(t') \text{ subject to } \min_{t'' \in \mathcal{P}(t)} \|t'' - t_{\text{in}}\|_2 \leq d_{\text{min}}$$

with the $L_2$ distance computed in CIELab space. After running this optimization, the optimally-ordered $t$ is returned. Though not technically necessary, permutations are included in both the objective and the constraints in order to reduce local minima. We optimize this function with Covariance Matrix Adaptation (CMA) [42], with $d_{\text{min}} = 35$ and constraints enforced by assigning very large penalties to themes that violate the constraints. CMA is run for 50 iterations with a sample size $N = 30$, taking approximately 5 minutes per theme. Optimal permutations are found by brute force enumeration. $d_{\text{min}}$ can also be varied for each color to enforce that certain colors be fixed (e.g., corporate colors) or allow more significant changes. Sample optimized themes are shown in Fig. 3.18.

Evaluation. MTurk users were shown the original theme and an optimized version and asked to select their preferred theme in an A/B test. Users could also select “neither” if they had no preference. Three sets of themes were used: the 50 worst-rated themes from the MTurk dataset, the 50 best-rated, and 100 random. We used 40 comparisons per task, and duplicates were added to identify and remove inconsistent users. After removal, the median number of participants per theme was 46. Two optimization tests were performed. In the first, themes were
Figure 3.18: Theme optimization. Left: original themes. Middle: optimized order. Right: optimized color and order. Regression rating $r(t_i)$ and mean ratings $r_i$ from a follow-up MTurk test.

Note in the fourth row where optimizing colors gives a lower rating than just re-ordering.
3.5.2 Theme Extraction

Designers often look to photographs for inspiration for selecting colors, and both Kuler and COLOURLovers provide tools for creating themes from images. Here we extract a color theme from an image \( I \). We extract themes with an objective function that attempts to represent or suggest an image while also being highly rated:

\[
\arg \max_t \alpha r(t) + \frac{1}{N} \sum_i \min_{1 \leq k \leq S} (\max \|c_i - t_k\|_2, \sigma) + \frac{\beta}{M} \max_k \sum_{j \in N(t_k)} \max (\|c_j - t_k\|_2, \sigma)
\]  

(3.6)

where \( c_i \) is a pixel color, \( t_k \) a theme color, and \( N \) is the number of pixels. The first term measures the quality of the extracted theme. The second term penalizes dissimilarity between each image pixel \( c_i \) and the most similar color \( t_k \) in the theme. Optimizing this term alone would be equivalent to \( K \)-means clustering with a modified distance function. The third term penalizes
dissimilarity between theme colors \( t_k \) and the \( M \) most similar image pixels \( N(t) \), to prevent theme colors from drifting from the image. We use \( M = N/20 \), \( \beta = 0.025 \), and \( \sigma = 5 \). We use the DIRECT algorithm for optimization [57], since it performs a deterministic global search without requiring a good initialization. Fig. 3.20 shows several examples including the original image, the extracted theme without the rating term \((\alpha = 0)\), and with \( \alpha = 3 \). For \( \alpha = 0 \), colors were sorted by value.

**Evaluation.** MTurk users were shown the original image and the extracted themes with and without the compatibility model, and asked to select their preferred theme (or neither if there was no preference). Tasks were structured as in previous tests, with a median of 35 participants comparing each image. Figure 3.21 shows that themes with the compatibility model are preferred for theme extraction than themes without the model \((p < 0.01)\).

Finally, follow-up work by Lin et al. [80] uses user data from MTurk to train a model for theme extraction. This model employs image and color features such as saliency, color coverage, color nameability, etc., and performs LASSO regression on the distance to human-rated themes. They compare with our dataset of images and themes, and find their approach produces improved theme extraction.

### 3.5.3 Color Suggestion

A common problem is to choose additional colors for a design, given some that have already been selected. Here, we consider a version of this problem in which we begin with a specific design taken from COLOURLOvers (Figure 3.22), and a specific theme created by COLOURLOvers users for this design. Four colors \( c_1, c_2, c_3, c_4 \) from this theme are fixed. We suggest candidates for the fifth color \( c \) so that the suggestions are compatible with the input colors. However, because the new color is assigned to a region, we also want the suggestions to contrast with neighboring regions.

To pick the first suggestion \( c^{(1)} \), we optimize:

\[
\text{arg max}_{c^{(1)}} \max_{t \in P([c_1, c_2, c_3, c_4, c^{(1)}])} r(t) \quad \text{subject to} \quad \|c^{(1)} - c_i\|_2 \geq d_i \quad i \in \{1, 2, 3, 4\}
\]

The constraint enforces that the new color contrasts with each previous color \( c_i \) by \( d_i \), which is computed as CIELab distance. \( d_i \) is set by the user to enforce scene-dependent constraints (e.g., a flower petal should contrast more with the background than a flower stem). Optimization is performed by brute-force search in color space.
Figure 3.20: *Theme Extraction*. **Left:** the original images. **Middle:** the extracted theme without compatibility model. **Right:** extracted theme with compatibility model. Creative-Commons photographs courtesy of Flickr users bombeador (Eduardo Amorim), marzinians (Dimitri Boisdet), szacharias (Stephen Zacharias), epsos, mikebehnken (Mike Behnken) respectively.
We produce a sequence of suggestions $c^{(j)}$ recursively. Specifically, when optimizing the $j$-th suggestion, we perform the same optimization as above, but add a constraint that the next suggestion be dissimilar from all previous:

$$||c^{(j)} - c^{(k)}||_2 \geq d \quad \text{for all } k < j$$  \hfill (3.8)

Fig. 3.23 shows an example of our model’s suggestions, as well as random sampling of colors satisfying the same constraints. The Kuler model was used as it lacks MTurk’s bias for brighter colors.

**Evaluation.** MTurk users were shown sets of color suggestions and asked to select the best and worst image from each set. Users were shown 24 sets, with each set consisting of 4 suggestions from our model, and 4 from random sampling, with all 8 randomly shuffled. 6 duplicate sets were included, in order to detect inconsistent users. A median of 64 people were used for
Figure 3.23: Color suggestions generated from an input design. Top: model suggestions. Bottom: random sampling from HSV space. The same set of constraints was used (see text) for both sampling methods. Color compatibility scores are listed for both sets, though not used for random sampling.
Our simple approach is limited in several ways. Our contrast measure is specified by a user on certain colours, rather than automatically computed. We also ignore the shape and semantics of regions, which plays an important role in design. For example, backgrounds are often light or dark, but rarely a mid-level lightness.

Follow-up work by Lin et al. [81] on design colouring addresses several of these problems. They use a data-driven approach using 8,200 coloured designs from COLOURLovers, similar to those in Fig. 3.23, to train a probabilistic model of design colouring. They compute a large number of region features, such as region size, spread, elongation, etc, which they use to learn distributions over color properties such as lightness and saturation. They also use our compatibility model as part of their final model.
3.6 Discussion

In this chapter we describe the first large-scale online studies of color preference, the first studies based on five color themes, and the first studies of user-generated color combinations. A number of observations emerge from this work. Designers creating themes do not uniformly sample the space of all possible themes. We find no support for the notion of hue templates as guides to aesthetics, except for the simplest, most basic themes. Instead, a number of simpler rules emerge, including a preference for a small range of colors, typically 2-3. The simplicity of a theme can be quantified by the hue entropy. The ordering of colors affects ratings. Certain color pairings are preferred, as is significant lightness variation, preferably a gradient. Segmentation of users by preference can lead to more accurate predictions. We also confirm previous findings that certain colors are preferred over others, but that preferences change in combination, which helps validate MTurk for studies of aesthetics.

Along with our observations on color preferences, we describe a learned model that can predict a rating for new color themes, and several demonstrations of tools that use this predictive model. Though our prototypes are only initial explorations, we hope that further refinement will yield tools that can help non-color experts navigate the sometimes daunting task of choosing colors.

Our model does have several limitations. First, the model ignores any individual user preferences for colors. While a model trained on mean ratings performs reasonably well, there is significant variation in individual preferences. Second, while we can analyze the 334 features weights, such analysis only provides clues about color compatibility. Learning weights on correlated features with an L1-norm is problematic as there is no guarantee which features will be selected. For example, if two features are identical, only one of the two will receive a non-zero weight. More thorough testing is required to determine which features are the most important, and to develop a simpler model which can generalize to an arbitrary number of colors. Finally, the model does not include any knowledge of content or how colors are used in practice. For example, in graphic designs, certain colors may be used as accents, or have associations with particular objects. However, we hope that our large-scale, evidence-based research for color aesthetics will inform future studies into color, and provide clues to better understanding color preferences. Modeling color trends and preferences over time is another area of future work, particularly since Kuler and COLOURLOvers continue to remain active websites.

This chapter provides an initial example of how data-driven machine learning approaches can aid understanding and development of new design tools. In the rest of this work, we use
different learning approaches to model layout and typography, but continue with the overall approach of leveraging data and large-scale, online studies to increasing our knowledge of aesthetics. Before that however, we first address one of the main limitations of this chapter, and turn to modelling individual color preferences.
Collaborative Filtering of Color Aesthetics

There is nothing outside of us that is not at the same time in us, and as the external world has its colors, the eye too, has colors.

– Johann Wolfgang von Goethe, *Theory of Colours*

Much of the real world is controlled as much by the ‘tails’ of distributions as by means or averages; by the exceptional, not the mean; by the catastrophe, not the steady drip; by the very rich, not the ‘middle class’. We need to free ourselves from ‘average’ thinking.

– Philip Anderson

Traditional theories of aesthetics focused on two opposing approaches: the objectivist approach which emphasizes object properties like symmetry and proportion, and the subjectivist approach which emphasizes individual and subjective responses to objects. In the previous chapter, we formulated an objectivist model for color aesthetics, investigating features such as hue compatibility and entropy, and training a model to predict average ratings for color themes. While that approach produced reasonable results, it ignored individual color preferences. In this chapter, we take a subjectivist approach, and propose a collaborative filtering (CF) approach for color aesthetics that models subjective user preferences.

In the previous chapter, a theme rating is the average of all user ratings, and a linear regression model is used to predict ratings. This objectivist approach of “averaging aesthetics” is stan-

---

A version of this chapter was published in the 2014 International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging (CAe) [110]. This paper was a collaboration between the author, Aseem Agarwala, and Aaron Hertzmann. The project page is at [www.dgp.toronto.edu/~donovan/cfcolor](http://www.dgp.toronto.edu/~donovan/cfcolor), and contains code and data used in this chapter.
Standard for predicting aesthetic ratings, and has been used for photographs [22, 90], paintings [78], and videos [100]. This approach is reasonable for several reasons. Firstly, it models the overall trend in aesthetics, which is often useful. Second, the learned models are often simple and interpretable; in the previous chapter we examined learned feature weights to analyze properties of highly rated color themes. Lastly, subjective preferences are extremely noisy, so averaging reduces noise which makes learning easier. However, this ignores any subjective variation in ratings due to personal preference. For color aesthetics in particular, subjective preferences are common. The wide variety of color palettes found in clothing and interior design speaks to the extent of individual color preferences. In this chapter, we use a CF approach to predict per-user ratings for color themes. We show that this approach outperforms an average rating model by a wide margin, indicating the usefulness of modelling individual aesthetic preferences.

One common approach to CF performs a matrix factorization on the rating matrix into latent vectors for users and items. However, this approach is limited in two respects. First, it ignores features of the items, which are often very important for aesthetic items like color themes or photographs. Items with very few ratings also benefit highly from features, as their latent vectors are underconstrained. Second, the model cannot predict ratings for novel items which are not present in the training data. To address these limitations, we use a feature-based approach based on probabilistic matrix factorization [135]. We extend this model to learn a latent linear transformation from features, instead of learning a per-theme latent vector. We then extend the model to handle nonlinear transformations using a neural network. Feature-based CF methods are not new, but we show that for visual aesthetics, a feature-based approach significantly outperforms the standard approach without features, and can predict aesthetic ratings for themes not seen during training.

Finally, we show that the learned model is useful for other aesthetic tasks. We use the model for measuring the distance between color themes, as nearby themes in the learned latent space have similar styles. For example, two themes with random hues may have a high pixel distance (as measured by comparing individual colors), but have a similar aesthetic style. An analogy with images would be red-eye removal: the before/after images are almost identical in pixel space, but have a high aesthetic distance. We also perform dimensionality reduction using t-SNE [160] to visualize the space of color themes. We show that the learned latent space improves the embedding, with similarly rated themes clustered together. More generally, our work is among the first to learn models of individual preference in visual aesthetics.
4.1 Collaborative Filtering

We next introduce the relevant research from the collaborative filtering literature. For a survey, see Su and Khoshgoftaar [146].

Matrix Factorization. Collaborative filtering often involves a set of ratings for items by users. One common approach uses latent factors, decomposing the rating matrix into the product of two matrices: a matrix $U$ modelling each user, and a matrix $V$ modelling each item. Salakhutdinov and Mnih [135] presented a simple probabilistic framework, later extended to a full Bayesian model [134]. The distance between latent vectors can also be used to model relationships between objects. For example, Latent Semantic Analysis [74] models the similarity between documents. We use this approach to model similarity between color themes.

Feature-based Collaborative Filtering. One limitation of simple factorization approaches is they ignore valuable features about items. For movies, the date, director, and country are all highly informative. Furthermore, they cannot generalize to unseen items since latent vectors are independent. While using item features has a long history in the collaborative filtering literature [98, 9], features are less common in matrix factorization approaches. Chen et al. [16] define a matrix factorization framework which uses item and user features, as well as global features. Our work is similar, but we present a probabilistic model for features which extends the PMF model of Salakhutdinov and Mnih [135], and also learns non-linear feature transformations. Adams et al. [1] also extend the PMF framework with Gaussian Process (GP) priors defined over the latent vectors using features. Our work also uses item features, but with a much simpler model: a single-layer neural network that learns a transformation from input features to latent features. Collaborative filtering problems often have tens of thousands of users and items, so GPs are problematic due to their large memory requirements.

4.2 Feature-based Matrix Factorization

4.2.1 Dataset and Features

To evaluate our proposed approach, we extend the MTurk dataset from the previous chapter. In this dataset, 13,343 color themes were chosen randomly from the Adobe Kuler website. Each theme was then rated on a scale of 1-5 stars by 40 participants on Amazon’s Mechanical Turk, producing a final dataset includes 528,106 individual ratings. A training set of 300,000 ratings was used, with a testing set of 128,106 ratings. A separate validation set of 100,000 ratings was
used to select model parameters. For features, we use the 334-dimensional feature vector described in the previous chapter.

### 4.2.2 Probabilistic Matrix Factorization

We first briefly describe the probabilistic matrix factorization model of Salakhutdinov and Mnih [135]. This approach uses a set of M items, N users, and integer rating values. \( R \) is a matrix of ratings, usually incomplete, where \( R_{ij} \) represents the rating of user \( i \) for item \( j \). We first define a latent vector for each user \( i \) and item \( j \) as \( U_i \) and \( V_j \) respectively, and model \( R \) as the product of the user and item latent vectors, plus noise. That is, each rating is defined as \( R_{ij} = U_i^T \cdot V_j + \epsilon \). The set of all user vectors is given by the matrix \( U \) (of dimension \( N \times K \)), and the item vectors as \( V \) (of dimension \( M \times K \)). The parameter \( K \) determines the size of the latent space.

We define the conditional distribution over the observed ratings as:

\[
p(R|U, V, \sigma^2) = \frac{1}{2} \prod_{i=1}^{N} \prod_{j=1}^{M} [N(R_{ij}|U_i^T V_j, \sigma^2)]^{I_{ij}}
\]  

(4.1)

\( N(x|\mu, \sigma) \) is a Gaussian distribution with mean \( \mu \) and variance \( \sigma \) and \( I_{ij} \) is the indicator function that is equal to 1 if user \( i \) rated item \( j \) and equal to 0 otherwise. Gaussian priors are also defined for \( U_i \) and \( V_j \):

\[
p(U|\sigma_U^2) = \prod_{i=1}^{N} N(U_i|0, \sigma_U^2 \mathbf{1}) \quad p(V|\sigma_V^2) = \prod_{i=1}^{N} N(V_i|0, \sigma_V^2 \mathbf{1})
\]  

(4.2)

MAP estimation is then used to learn the latent vectors for items and users. The log posterior of Eqns 4.1 and 4.2 is used to define a sum-of-squared-errors objective function:

\[
E(U, V) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T V_j)^2 + \alpha_U \sum_{i=1}^{N} ||U_i||_{Fro}^2 + \alpha_V \sum_{j=1}^{M} ||V_j||_{Fro}^2
\]  

(4.3)

The gradients with respect to \( U_i \) and \( V_j \) are simple to compute, and training done by gradient descent; see Salakhutdinov and Mnih [135] for details. In our experiments, we set the dimensionality of the latent space to \( K = 5 \) based on a validation set of ratings, described in the next section.

### 4.2.3 Linear Feature-based Matrix Factorization

A major disadvantage of the previous approach is it ignores item information which could help rating estimation. For color themes, or other visual stimuli, this information is important for
Figure 4.1: Factorization models. (a) standard probabilistic matrix factorization learns latent vectors for each user $U$ and each item $V$. (b) feature-based matrix factorization learns a linear transformation $T$ from fixed item features $F$ to the latent space. (c) features are transformed to the latent space using a neural network. (d) both fixed latent features and nonlinear feature transformations are used to model users.

prediction. Another disadvantage is a lack of generalization; ratings for new items not present in the training data cannot be estimated.

Given a feature vector $F_j$ for each item $j$, we can learn a mapping from feature space to latent space. We first present a linear transformation $T$ of the feature vector:

$$E(U, T) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T \cdot (TF_j))^2 + \alpha_U \sum_{i=1}^{N} \|U_i\|^2_{Fro} + \alpha_T \sum_{k=1}^{K} \|T_k\|^2_{Fro} \quad (4.4)$$

The gradients with respect to $U$ and $T$ are again straightforward, and training is done with gradient descent. The matrix $T$ is of size $Q \times K$, where $Q = 334$ and $K = 15$.

4.2.4 Nonlinear Feature-based Matrix Factorization

We can also define a nonlinear transformation function $T(F_j; W)$ with parameters $W$:

$$E(U, W) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T \cdot T(F_j; W))^2 + \alpha_U \sum_{i=1}^{N} \|U_i\|^2_{Fro}$$

The nonlinear transformation is a neural network trained using back-propagation. When learning the parameters, the user vectors $U$ and the network parameters $W$ are updated alter-
nately; the parameters are fixed for one set and the gradient calculated for the other. The users’ latent vectors are updated at each iteration as before. The users’ latent vectors act as a final layer of linear weights on the neural network, with the errors are back-propagated through the network to $W$.

There is also no prior on the network weights. While such regularization is trivial to add, we expect our simple model shared over our hundreds of thousands of datapoints will be robust to overfitting. Therefore, sparsefying or penalizing higher weights may rule out good transformations. In practice, small weights are learned with little overfitting (Fig. 4.2). Initial experiments with regularization revealed no improvement. The neural network included 200 logistic units, and the dimensionality of the latent space was $K = 15$, set using validation ratings. We found the performance of the FPMF models were fairly robust to parameters changes, and results did not change significantly.

A further extension is to add feature-based latent vectors for users as well. Each user in our dataset self-reported their gender, experience, country, and age. We can therefore use these binary features with a second transformation:

$$E(U, W_1, W_2) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} \left( R_{ij} - [U_i T_i(G_i; W_1)]^T \cdot T_2(F_j; W_2) \right)^2 + \alpha_U \sum_{i=1}^{N} \|U_i\|_{\text{Fro}}^2$$

Where the vector $[U_i T_i(G_i; W_1)]$ is the concatenation of the user’s latent vector $U_i$ with the output the neural network $T_i$ given the user features $G_i$, and parameters $W_1$. We use 200 logistic units for the item-feature network, and 50 logistic units for the user-feature network. $U_i$ has dimension 15, and $T_i$ has dimension 5.

### 4.3 Experimental Results

The first baseline we compare our CF approach against is an average aesthetic model. We use the LASSO algorithm of previous chapter: linear regression with an L1-norm $|\cdot|_1$. This model is trained on the average of all training ratings for a theme. However, testing is done by evaluating each individual rating, not the average. This baseline indicates how much individual user preferences affect the rating.

We also compare our feature-based models with regular PMF to evaluate how important features are for modelling visual aesthetics. As mentioned earlier, one important advantage of feature-based models is the ability to handle test themes not seen in training. We therefore test
Table 4.1: Model testing. We evaluate various models using the RMSE of test theme ratings. 'Averaged' is a linear regressor trained on mean theme ratings (Sec. 3.3). Nonlinear FPMF (V) uses a neural network with theme features. Nonlinear FPMF (U+V) uses a neural network with user and theme features. ‘Seen’ and ‘Reduced’ include previously seen users and themes. The ‘Novel’ set has no themes used in training. ‘Reduced’ uses only theme features (the theme’s CIELab color values); ‘Seen’ and ‘Novel’ sets use the full 334-dimensional feature vector.

<table>
<thead>
<tr>
<th>Method</th>
<th>Seen</th>
<th>Reduced</th>
<th>Novel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Averaged</td>
<td>1.082</td>
<td>1.107</td>
<td>1.081</td>
</tr>
<tr>
<td>PMF</td>
<td>0.964</td>
<td>0.964</td>
<td>-</td>
</tr>
<tr>
<td>Linear FPMF</td>
<td>0.842</td>
<td>0.969</td>
<td>0.841</td>
</tr>
<tr>
<td>NL FPMF (V)</td>
<td>0.831</td>
<td>0.945</td>
<td>0.829</td>
</tr>
<tr>
<td>NL FPMF (U+V)</td>
<td>0.829</td>
<td>0.944</td>
<td>0.828</td>
</tr>
</tbody>
</table>

on a dataset (‘Novel’) where all test ratings are for new themes. Feature creation can be time-consuming and often requires expert knowledge. We therefore also explore the models’ performance with a reduced feature set. With a smaller set of features, we expect nonlinear FPMF should perform better than linear FPMF, as the nonlinear transformation should compensate for less hand-crafted features. We therefore test a feature set (‘Reduced’) with only the 15 CIELab colors, and compare to the full 334-dimensional feature vector.

Table 4.1 shows our main result: the error for the average predictor is substantially higher than those which model individual user preferences. We also show the value of using features when modelling visual aesthetics, as the feature-based FPMF model performs much better than PMF at predicting theme ratings. Nonlinear FPMF also out-performs linear FPMF, with better relative performance with fewer features. However, adding demographic user features only gives a very small improvement. Fig. 4.2 plots the error on the validation set, and also shows how the PMF model overfits the data, unlike the FPMF models.

We next investigate the effects of user modelling and demographic features. As a baseline, we trained the nonlinear FPMF model with a constant \( U_i \) for all users and no demographic features, using the ‘Novel’ dataset. This model gave a RMSE of 1.079, as compared to 0.828 for the nonlinear FPMF with user modelling, again demonstrating its value. Note that this approach closely matches the RMSE of 1.081 for the averaged LASSO predictor, which also ignores user modelling.

We then tested using only demographic features to model the user. Specifically, we removed
the latent vector $U_i$ from Eqn. 4.5, and modelled users only by the neural network $T_i(G_i)$, where $G_i$ are the demographic features of user $i$. This model produced a RMSE of 1.066, suggesting that while demographic features are informative, they are far less important than modelling individual preferences for color themes. The marginally better performance with demographic features is slightly surprising. Previous research on webpage aesthetics, including colourfulness, found significant differences between demographic groups [128]. One reason may be that Reinecke's dataset included a broad sampling of countries, whereas the vast majority of the MTurk color dataset are from USA or India. It is also likely that color theme preferences have more variation within groups than across them, particularly compared to webpage aesthetics.

In Fig. 4.3, we show a concrete example for two users with different aesthetic styles. We show highly and poorly rated themes for the two users, along with predicted ratings for new themes using the nonlinear FPMF model, demonstrating that our CF model accurately captures the users’ aesthetic preferences. Our method can also predict ratings distributions, by predicting ratings for all users in the training set. In Fig. 4.4, we show the distribution for two novel testing themes not seen in the training data. This figure shows that there can be large differences between distributions; the variance of the top theme is much higher, indicating more disagreement in ratings than the bottom theme.

### 4.4 Applications

Navigating the space of color themes is a difficult problem with little previous work. User-specified tags are often used for searching similar themes (e.g., “pastel,” “venice,” “stone,” “rose”)
4. Collaborative Filtering of Color Aesthetics

Figure 4.3: Collaborative filtering example. The left three columns show highly and poorly rated themes from two users with different aesthetic preferences. The ratings for the two users are denoted as \( r_i \) and \( r_j \). The right two columns shows our predicted ratings \( p_i \) and \( p_j \) for new themes using nonlinear FPMF. Predicted ratings may be below 1 or above 5, as the above example shows, since the prediction is the product of latent vectors for the item and user. In practice, these values would be clipped to the range (1-5).

Figure 4.4: Predicting rating distributions. Given a novel theme (i.e., one not present in the training set), we predict the ratings for all users in the training set, and plot the distribution of their ratings. In this example, both themes have a mean rating of 3.00, but the left theme has greater disagreement (std. dev. of 1.03 vs 0.64).

but this approach is limited. The main problem is the lack of a distance metric for themes. We wish to find ‘similar’ themes, but similarity is poorly understood for color combinations. One simple solution is to take the sum of color differences in a perpetually uniform color space like CIELab. However, this naive approach does not model the relationships between colors, or the overall style of the theme. For example, a color theme which lies along a gradient (e.g., dark to light) should be closer to the flipped theme (i.e., light to dark) than to a random permutation of the colors which does not preserve the gradient, though it may have a lower CIELab distance. An
analogy for images would be before and after red-eye removal. While both images are extremely similar in pixel distance, they have a large aesthetic disparity.

Instead of color differences, we propose a similarity metric for color themes which measures differences in aesthetic style. Specifying such a distance is not intuitive. However, theme ratings can be used as a proxy for measuring aesthetic distances, as themes which are aesthetically similar will tend to have similar ratings. We would like a transformation for themes such that, in this new space, a small distance results in a small rating difference. FPMF produces such a transformation, incorporating aesthetic differences and grouping similarly rated themes. Similar latent factor approaches have been used to detect synonyms [74], and to visualize similar movies [65].

In Fig. 4.5 we show several themes with a large CIELab distance but a small distance in the FPMF latent space, and vice-versa. Since the scales of the spaces are different, we also report the distance sort order in each space. That is, for each theme, we first calculate the distances to every other theme. These distances are then sorted, and the order number reported. A value of 0 indicates the second theme is the closest theme to the first in this space. A value of 1 is the most distant. This metric gives a relative sense of the distances.

In the top three examples, we show themes which are visually quite similar but have a large CIELab distance. By contrast, the latent distance is much smaller. In the top example, the two hues are switched; in the second, the gradient is reversed; in the third, both themes are poorly ordered with bright primary colors. In the bottom three examples, themes with a small CIELab distance are visually quite distinct, which is reflected by a larger latent distance. A naive CIELab distance does not account for contrast between colors. If two themes have similar lightness and saturation, they will have a fairly low CIELab distance. However, a single modified hue can greatly decrease the perceived similarity and aesthetic rating.

To visualize the space of color themes, we use t-SNE [160] to create a 2-D embedding. In Fig. 4.6, we compare an embedding using the CIELab distance with one using the latent vectors (please zoom in for more detail). While both embeddings lack clear clusters, the results are improved with FPMF in several ways. First, there is an overall diagonal light to dark trend with FPMF not present in the CIELab embedding. Second, bright themes with significant color variation (e.g., the third theme of Fig. 4.5) are clustered in the top right whereas they are spread out in the CIELab embedding. Particular hues are also better grouped (e.g., the blue theme of Fig. 4.5). We also plot the embeddings with the mean user ratings in Fig. 4.6 (bottom). This figure shows that similarly rated themes are being placed closer together using the latent vectors; with CIELab distances, poorly rated themes are spread throughout.

We can also use t-SNE to visualize users. Fig. 4.7 shows a 2D embeddings of users’ latent
Figure 4.5: Distances between themes. **Top:** themes with a large CIELab distance but small FPMF latent distance. **Bottom:** themes with a small CIELab distance but large latent distance. The (d)istances are computed in CIELab and FPMF. Note that distances in CIELab and FPMF are not directly comparable as they are different spaces (i.e., a distance of 1 is not equivalent in both spaces). To compare the different spaces, we report the sorted distance (o)rder for all themes. 0 indicates no other theme is closer, 1 indicates no theme is farther away. These results indicate the latent features are better for measuring visual similarity.
4.4. Applications

Figure 4.6: t-SNE embedding of 2000 color themes. Top left: Embedding of CIELab color values. Top right: Embedding of FPMF latent features. Bottom left: Mean user ratings for each theme (CIELab embedding). Bottom right: User ratings of FPMF embedding. Please zoom in for detail. The FPMF embedding clusters similarly rated themes better than the CIELab embedding.

vectors, coloured by demographic features. The figure does show some degree of clustering, indicating users with similar preferences. Some clusters are predominately of one country, though the map fails to show a clear separation between users of different countries. There is also little separation of users based on their gender. We also tried labelling the users by their age, but there was similar degree of inter-group variation. These findings reinforce the claim that differences in color aesthetic preferences between demographic groups are much lower than differences within the groups.
Figure 4.7: t-SNE embedding of users with country-of-origin, gender, and age labels. While there is some clustering of user preferences, there is substantial variation within the demographic groups.
4.5 Discussion

Modeling aesthetic preferences is an exciting area with many potential applications from music, to image processing, to fashion and design. Large-scale datasets also offer the opportunity for greater understanding of aesthetic preferences. To our knowledge, collaborative filtering approaches have not been explored previously for modelling aesthetic ratings. Previous approaches average over all ratings to measure an overall aesthetic score. This approach is appropriate when no information is available about a new user. However, when previous information is available, modelling individual user preferences can achieve significantly better performance than average aesthetic models.

In this chapter, we use a feature-based probabilistic matrix factorization (FPMF) model to predict individual user ratings. We introduce two simple extensions to the original PMF framework. First, instead of solving for a latent vector for each color theme, we solve for a transformation from theme features to the latent space. Second, we propose a nonlinear transformation within the factorization using a neural network. We show a feature-based approach significantly outperforms one which ignore features. Image and music aesthetics are possible domains for the FPMF model as both have large datasets and rich feature sets.

We also show the model’s usefulness for understanding and visualizing color themes. Latent factor transformations can measure the aesthetic distance between themes which can be difficult to specify directly. We also use this representation to visualize the space of color themes. Given the vast datasets of color themes and images available online, building interfaces which use aesthetic models to help navigate these spaces is an exciting area of research.
5

Learning Layouts for Single-Page Graphic Designs

Design is unity out of multiplicity, by selection, limitation, formation, rhythm, via static or directedly moved balance, via a system.

– Kurt Schwitters, Die neue Gestaltung in der Typografie

In this chapter, we move from simple linear models of color compatibility, which were mostly stripped of a design context, to a more practical and complex model of graphic design layout. Designs are often difficult to create, as they must clearly convey information while also satisfying aesthetic goals. Designers must now also create designs for a wide variety of display sizes, from mobile phones to posters, and must often retarget designs from different sizes. Furthermore, many designs are created by inexperienced users with little training in graphic design. Automatic tools for creating, adapting, and improving graphic design layouts could greatly aid designers, and in particular, novice users.

This chapter considers an important class of designs: single-page graphic designs such as advertisements, fliers, or posters (Figure 5.1). These designs often consist of a small number of text and graphical elements. While there is previous work on automatically creating web pages and article-type designs, there is little research on generating single-page graphic designs. These designs are challenging as they are less structured and have a wider range of sizes and

A version of this chapter was published in IEEE Transactions on Visualization and Computer Graphics, 2014. [111]. This paper was a collaboration between the author, Aseem Agarwala, and Aaron Hertzmann. The project page is at www.dgp.toronto.edu/~donovan/layout. This page contains examples to supplement the chapter, including more importance maps examples, all training data for our models, as well as designer and crowdsourced layout examples.
We present a model of single-page graphic designs based on design principles such as alignment and balance. The model can synthesize layouts in various styles learned from examples, retarget layouts to different sizes, and improve designs based on design principles.

Automating single-page graphic design is a complex and unsolved problem. In this chapter, we focus on the layout problem: specifying the locations and sizes of design elements. We assume a set of text and graphical elements are provided as inputs along with associated metadata, such as the number of lines for text elements. Our goal is to output a visually pleasing arrangement of elements in a particular style. Modeling layout (i.e., element position and scale) is an important step towards formalizing the difficult problem of design.

We present a new energy-based model for evaluating layouts based on graphic design principles and stylistic goals. Due to the complexity of graphic designs, our approach has two stages. Given a design, we first perform a novel analysis stage that infers hidden variables corresponding to perceptual properties such as perceived importance, alignment, and grouping. These variables are then used as part of the model to evaluate a design. For example, the system analyzes perceived element alignment in a layout, and then an energy term penalizes misalignments. Given this model, we use optimization to synthesize a new layout. Our model has a large number of parameters, and we show how to learn their values from one or more examples using
Nonlinear Inverse Optimization (NIO) [82].

We apply our model to three applications. First, the system can synthesize new design layouts in different styles learned from examples. Second, given an existing design, the system can retarget the design to a new size and/or aspect ratio. Lastly, we demonstrate a “design checker” that can improve an existing design to better match basic principles of graphic design. Model parameters are learned independently for each application using NIO and a few examples. This initial system is designed for quality rather than efficiency, and is too slow for interactive applications. We investigate interactive design interfaces in Chapter 6.

To evaluate our method, we automatically generate, retarget, and improve a number of designs. We compare our retargeting results with designs created by a professional designer, and also with novice users performing the same tasks on Mechanical Turk. We show that our results are generally as good as those produced by the average human in our crowdsourced study, though not at the level of a professional designer. Graphic design is a particularly hard task to automate. Even humans are not particularly good at it, apart from designers who often have significant experience and specialized training. By contrast, classic problems like speech recognition and computer vision are performed well by most humans. Nonetheless, automatically producing results on par with untrained humans represents significant progress on this unsolved problem.

In this chapter we return to an objectivist model of aesthetics, specifying energy terms which capture properties of a design. However, rather than emphasizing a single “correct” style, we learn parameters from examples to capture a variety of styles. Learning subjective mappings from user ratings to model parameters would be straightforward, and is left for future work.

5.1 Overview

This work defines a graphic design as a set of visual elements, including text and graphics, represented as images with associated metadata. We focus on the layout problem: determining the positions and scales of these elements, denoted X. Our overall goal is to create layouts which respect the principles of graphic design, such as alignment and symmetry in a variety of styles. Our main contribution is an energy-based model \( E(X; \theta) \) which evaluates a layout \( X \) using parameters \( \theta \) which define the desired style and intentions of the user (Section 5.2).

Due to the complexity of the problem, we use a multi-stage approach. To evaluate the energy of a layout, the system first estimates hidden variables (h) that correspond to important visual properties. The hidden variables are the perceived importance of each element, a grid-based
segmentation of the layout, and labels specifying alignment groups (Section 5.3). We then define our energy terms in terms of these hidden variables as $E_h(X, h; \theta)$. The energy function enforces many other design principles, including alignment, symmetry, and white space (Section 5.4).

Given this energy function and a suitable optimizer for $X$ (Section 5.5), the system can generate design layouts in a variety of styles. Due to its high dimensionality, an important goal is learning $\theta$ without requiring time-consuming manual parameter tuning. We use Nonlinear Inverse Optimization (Section 5.6) to learn parameters $\theta$, which are used to generate layouts for new designs.

We present applications of the model, including results and evaluation, in Section 5.7. First, we demonstrate layout synthesis, where layouts are generated for designs in a variety of styles with learned parameters. We then show design retargeting results, where a previous layout $X_p$ is modified for a different output size. We also show results of design improvement, which takes an existing layout and optimizes it to enforce design principles.

## 5.2 Graphic Design Layout Model

We measure the overall quality of a layout as a weighted sum of energy terms:

$$E_h(X, h; \theta) = \sum_i w_i E_i(X, h; \alpha_i, X_p)$$  \hspace{1cm} (5.1)

A design layout $X$ is defined as the $x$ and $y$ positions, height, and alternate ID of each element. Alternate IDs select between different alternate elements, usually text blocks with different internal alignments. The energy terms $E_i$ are defined in Sec. 5.4. $h$ are the hidden variables described in Sec. 5.3, and $\theta$ are the model parameters. When the model is used for design retargeting or improvement, the user provides a previous layout $X_p$.

The inputs to the model include the design elements, metadata for each element, an output width and height, and optionally a previous layout $X_p$. Given the inputs and the model, our system optimizes $X$ to synthesize a design layout. By changing the parameters $\theta$, we can generate layouts in various styles.

Elements are provided as images, along with user-defined metadata. The system only requires three metadata values for each element: the class (text or graphics), the number of lines (for text elements), and an importance value (low, medium, high, or very high). Optionally, if the element is part of a group, a group ID may be provided. The user may also provide binary masks to specify if a person or face is present in the graphic, or an important region which
cannot be obscured by text. However, these masks are not required nor must they be precisely drawn.

Model parameters are divided into two groups, $\theta = [w, \alpha]$. Each energy term $E_i$ has a positive weight $w_i$, and most terms have a nonlinearity parameter $\alpha_i$. The weights are constrained to be positive. For example, a positive weight for the misalignment energy term means the model can only encourage alignment. We also include some reversed energy terms. For example, one term encourages symmetry, and a reversed term encourages asymmetry. The sum of these two nonlinear terms, both with positive weights, allows the model control over the preferred amount of symmetry.

We use a sigmoid function in many of our energy terms: $S(x; \alpha) = \arctan(x\alpha) / \arctan(\alpha)$. We often use it to reshape energy terms as $S(E_i(X); \alpha)$, as large values of $\alpha$ make the energy more sensitive to small changes, such as in styles with little white space between elements.

5.3 Design Analysis/Hidden Variables

Before evaluating a layout, the system performs an analysis stage to infer hidden variables corresponding to how a human viewer perceives the layout. We infer three key variables: the perceived importance of each element, labels specifying element alignment, and a grid-based segmentation. See Figure 5.2 for an example. These perceptual properties cannot be provided initially as metadata or a document structure; they are a function of the arrangement of elements on the page. This section explains our approaches for computing these hidden variables. Section 5.4 explains how the variables are used in the energy function.
5.3. Design Analysis/Hidden Variables

Figure 5.3: Importance maps. Given a design (top left), MTurk users mark what they consider important. While the individual maps are noisy, when averaged over 20-30 users (bottom right), the mean maps are reasonable. Design courtesy of Flickr user Dániel Perlaky.

5.3.1 Importance Map

When creating a design, controlling the perceived importance of various elements is crucial, and designers often arrange elements to convey their importance [165]. Color, size, and location all contribute to an element’s perceived importance, but formulating a mathematical formula is difficult. There are obviously relative differences in importance: a large graphic in the center of the design is far more important than a small URL in a corner. But how does location affect importance generally? How does the importance depend on other elements?

Crowdsourced Design Importance. Inspired by Judd et al. [58], we model importance using a data-driven approach. First, we collected 1,075 graphic designs from Flickr. We then performed an MTurk study asking 35 users to label important regions in a design, and averaged the responses over all users. The system then computes per-pixel features, described below, and trains a linear regression model on the mean importance map. This model is then applied to new designs to predict the importance of elements.
MTurk Importance Study. To gather importance data, we performed an MTurk study asking users to label important regions in a design. We did not provide an explicit definition of importance, but instead showed 7 example labelings. 1,075 graphic designs were shown to 35 MTurk users who labeled the most important regions. Users were paid 30¢ for labeling each set of 20 designs, and 758 MTurk users completed the study. Duplicate designs were added randomly and inconsistent users were removed.

The individual MTurk importance maps are often quite noisy, with significant variation between users. However, averaged over many users (20-30), the mean importance maps often give a plausible ranking of importance. See Fig. 5.3 for examples of individual user’s importance maps, Fig. 5.5 for several mean importance maps, and the project page for more examples\(^1\). This result is surprising, because the mean importance should not necessarily create a relative ranking of elements; if everyone performed the task the same, there would be no relative difference.

We hypothesize that each element has an unknown importance rank, and that people choose different numbers of elements to label. Some users label only the most important element, others only a few, others most of the elements. If each person marks the top \(k\) elements, for an individual value of \(k\), we will produce the correct ranking by averaging the “votes” of all users.

Importance and Eye Tracking Importance is related to image saliency \([115, 49, 58, 35]\), which is usually equated to predicting eye fixations. However, eye fixations can be distinguished from image importance in several ways. Eye fixations include significant noise, and often appear in unimportant regions as the user scans the design. For example, fixations vary significantly over text blocks, even though the text’s importance is uniform. Importance also depends on a viewer’s interpretation of the design. When a user visually parses a design, they segment regions, detect objects, and infer relative importance. For example, in the bottom left image of Fig. 5.3, a user has coarsely segmented the two figures and three blocks of similar text size. In the top row, a user segmented and labelled only the title as important.

From a practical standpoint, eye tracking studies require expensive hardware and are time-consuming to perform. By contrast, importance data can be easily gathered from crowdsourced studies. Given these limitations, we use importance for our design model. However, eye fixations are potentially useful for understanding the perception of designs. Ideally, the order of low-level eye fixations should match a high-level importance map. Eye tracking could reveal distracting graphics or fonts, or poor layouts where the read order is unclear.

\(^1\)www.dgp.toronto.edu/~donovan/layout
Figure 5.4: Crowdsourced features. We use crowdsourcing to extract high-quality features including people and face detection, and text size. Designs courtesy of Natasha Mileshina and Ben Keenan.

Features. To learn importance maps, we calculate per-pixel features to train a linear regression model. Features include RGB color channels, efficient low-level saliency models [115, 49], and multi-scale contrast features [84]. We include global position features including the distance to the image center, boundaries, diagonals, and Third lines, and the intersections of the Third lines.

Our features also include labeling of people, faces, and text. Because existing methods for face and person detection often fail for graphic designs, we use crowdsourcing to find these labels. For our training data, MTurk users drew binary masks for both people and faces and the average for each was taken over approximately 30 users. In a separate task, workers marked text blocks along with the number of lines in each block. See Fig. 5.4 for examples. The system computes per-pixel features including the fraction of labels over all users, at least one user labeling, the user count, and the mean labeling over the entire image. The connected components of the label maps are computed and the segment size (both absolute and relative to the largest segment) and the number of segments are used as features. Text features include the size (both absolute and relative to the height), and number of lines.

When creating features, efficiency is a concern since the model is used as part of the energy
5. Learning Layouts for Single-Page Graphic Designs

<table>
<thead>
<tr>
<th></th>
<th>Fixed</th>
<th>IK</th>
<th>HZ</th>
<th>J</th>
<th>G</th>
<th>Full</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>0.304</td>
<td>0.295</td>
<td>0.253</td>
<td>0.280</td>
<td>0.251</td>
<td><strong>0.155</strong></td>
<td><strong>0.165</strong></td>
</tr>
<tr>
<td>$R^2$</td>
<td>0</td>
<td>0.054</td>
<td>0.306</td>
<td>0.318</td>
<td><strong>0.739</strong></td>
<td><strong>0.702</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of saliency and importance models. Saliency Models: (IK) Itti and Koch [53], (HZ) Hou and Zhang [49], (J) Judd et al. [58], (G) Goferman et al. [35]. (Full/Fast) our importance modelling approaches.

function for optimizing designs. A feature set which requires many seconds to compute is not practical. However, we also experimented with more time-consuming features including other saliency detectors [53, 58, 35], steerable pyramid filters [143], and object detectors [27] for more accurate modelling of importance. We next describe results using both the fast and full model. In practice, we found the fast feature set worked well, and was used for the model in the following sections.

Note that these crowdsourced features are used only for training the importance model. When we apply the model on synthesized layouts (as in Fig. 5.2), text location and size are known during synthesis, and person and face label images can be created by a single user. We do not require MTurk labels when synthesizing designs.

**Prediction.** The model computes features $y(p)$ for each pixel $p$ and predicts importance $r(p)$ using a linear regression of the features: $r(p) = w^T y(p) + b$. Parameters $w$ and $b$ are learned with LASSO [155], using $L_1$ regularization:

$$
\text{arg min}_{w,b} \sum_i (w^T y_i + b - r_i)^2 + \lambda \|w\|_1
$$

where $r_i$ is the mean importance from MTurk users. The optimal parameters $w$ and $b$ are computed via a convex optimization [29] (glmnet package), with $\lambda = 0.00037$ selected by cross-validation.

Table 5.1 compares our two models and existing image saliency models, using 10-fold cross-validation with a 0.9/0.1 random train/test split of our 1,075 designs, with 1,000 pixels randomly sampled from each image. Fig. 5.5 shows a few example designs from the test set. We report the root-mean-square error (RMSE) for our predictor, as well as the $R^2$ coefficient where 1 is a perfect predictor and 0 is the baseline of simply predicting the mean importance value $\bar{y}$.

$$
R^2 = 1 - \frac{\sum_i(y_i - f_i)^2}{\sum_i(y_i - \bar{y})^2}
$$

(5.3)
Figure 5.5: Design importance. Given an design from a test set, we show the mean MTurk importance map, the saliency model of Goferman [35], and our full and fast importance models. The fast model is used in all following examples. Designs courtesy of William Berry, Dániel Perlaky, and Ben Keenan.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Text</th>
<th>FP</th>
<th>Sal</th>
<th>TFP</th>
<th>G+TFP</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>0.223</td>
<td>0.297</td>
<td>0.231</td>
<td>0.200</td>
<td>0.175</td>
<td>0.155</td>
</tr>
<tr>
<td>( R^2 )</td>
<td>0.462</td>
<td>0.045</td>
<td>0.426</td>
<td>0.569</td>
<td>0.669</td>
<td>0.739</td>
</tr>
</tbody>
</table>

Table 5.2: Feature set comparison. (Text) Text, (FP) Face and Person, (Sal) All Saliency, (TFP) Text, Face, and Person, (G+TFP) Goferman [35] and Text, Face, and Person, (Full) All Features.

These results show that existing image saliency models poorly predict the human-created importance maps. This result is unsurprising since these methods predict eye fixations, not im-
portance. Existing saliency methods are also designed for natural images, so they fail to capture text importance. Both our feature sets work quite well at predicting human importance, with the full model performing slightly better.

To evaluate the extent that particular features contribute to perceived importance, we trained the regression model with different subsets of features. Table 5.2 compares the results using text, face and person, and saliency features. We also combined our crowdsourced features with the best-performing saliency measure [35]. These results suggest that text features and salience both play a large role in visual importance, which is intuitive since designers often make important elements more salient or eye-catching.

### 5.3.2 Alignment

The alignment of elements is crucial to how viewers perceive a design. Our system performs an analysis stage which labels all aligned elements as well as alignment groups. Elements with slight misalignments are also labeled as aligned, as the model will later penalize them. We define 6 possible alignments types: Left, X-Center, Right, Top, Y-Center, and Bottom. Alignment indicator variables between elements $i$ and $j$ are denoted $I^a_{ij} \in 0,1$.

We use simple heuristics to compute alignments using element bounding boxes. First, the difference in the bounding box edges or center positions must be below a threshold:

$$A^a_{ij} = (d^a_{ij} < \tau_{\text{align}})$$

where $a$ is the alignment types, $d^a_{ij}$ indicates the distance between two elements $i$ and $j$ depending on the alignment type using element's bounding boxes. For example, if $a=$Left then $d^{\text{left}}_{ij}$ measures the difference in the left edge of the bounding box for both elements. The threshold was set to $\tau_{\text{align}} = 0.065$ where the units are pixels, normalized by design size. Note that slightly misaligned elements are still labelled as aligned, allowing slightly misalignments to be penalized by the energy model.

Secondly, elements may not align if another element lies between them:

$$B_{ij} = (b_{ij} < 1)$$

where $b_{ij}$ is the number of elements between $i$ and $j$.

Lastly, if a text block is internally aligned, it may only align with other elements with that alignment type. For example, a left-aligned text block may only left-align with another element. We denote the internal alignment indicator variable as $N^a_i$. Single-line text and graphical elements can align to any type. Fig. 5.6 illustrates the alignment labeling graphically.
5.3. Design Analysis/Hidden Variables

Figure 5.6: Element alignment labeling. We show the x-axis alignment labels for a set of center and nearly left-aligned elements. Elements align if the differences between their bounding box edges or centers are less than a threshold. Note that the misaligned elements on the right are still labelled as aligned; the model will then penalize these misalignments. Elements do not initially align if other elements lie between them. For example, elements a and c, and d and f are not initially aligned. These elements are then combined into an alignment group so that $I_{ac}^a_{X\text{-Center}} = 1$ and $I_{df}^{\text{Left}} = 1$.

The alignment labeling is the conjunction of these terms:

$$I_{ij}^a = A_{ij}^a \land B_{ij} \land N_i^a \land N_j^a$$  \hspace{1cm} (5.6)

Within each axis, two elements may normally align by only a single type. The indicator variable with the minimum alignment distance $d_{ij}^a$ is set to 1, and the other two are set to 0. However, if all types align perfectly ($d_{ij}^a = 0$ for all types) then all three indicators are set to 1.

We next define an alignment group as a connected set of aligned elements. If elements $i$ and $k$ are aligned, and elements $k$ and $j$ are aligned by the same type, then $i$ and $j$ are set to aligned ($I_{ij}^a = I_{ik}^a \land I_{kj}^a$). We denote group membership using binary indicator variables $I_g^a$. See Fig. 5.6 and Fig. 5.7 for examples.

5.3.3 Hierarchical Segmentation

Designers often use grids or rectangular regions to organize elements. A viewer perceives this structure and relates alignment, grouping, and symmetry to these regions. Our system estimates this layout structure and calculates energy terms based on it. The algorithm takes as input the layout, binary masks for each element, and element classes (graphic or text); the output is a hierarchical segmentation of the design into non-overlapping rectangular regions.
Figure 5.7: Alignment groups. Rectangle colors indicate the detected alignment groups, with the orientation and position of the rectangles indicating the alignment type. Coloured connector lines groups the elements, with the deviation of the rectangle from the connector line indicating the misalignment. The left design illustrates a perfectly aligned group (in blue) and misalignment of the bottom and center elements (in red and green). The right design illustrates that elements cannot group with inconsistent internal alignments (e.g., the center-aligned address), and that elements cannot group with unaligned elements between them.

The proposed algorithm segments a design by vertically or horizontally splitting regions which contain both text and graphics. Each split is evaluated using a cost function which measures the intersection of the split with elements, the separation between graphic and text elements, and distance to the region center. The algorithm recursively segments regions until a region contains only elements of the same class, or a user-specified maximum depth is reached. Lastly, empty or adjacent regions with the same element classes are merged.

Objective Function. Given a rectangular region \( r \), a cut \( c \) is defined as an \( x \) or \( y \) position in \( r \) which splits the region into two rectangular sub-regions \( r_1 \) and \( r_2 \). The system evaluates cuts based on three simple criteria. First, cuts should not be placed over elements, especially near an element’s center. One energy term penalizes cuts based on the distance to each element’s bounding box. Cuts nearer the region boundaries pay a low cost, while cuts near the center pay a high cost:

\[
F_{\text{int}}(c) = \frac{1}{n} \sum_{p \in c} \max_i (I^i_p \delta_i^c(p))^2
\]  

(5.7)

where \( p \in c \) are the pixels \( p \) along the cut \( c \), \( I^i_p \) is an indicator variable indicating if element \( i \) overlaps with pixel \( p \), and \( \delta_i^c(p) \) is the distance of pixel \( p \) to the bounding box of element \( i \). This distance depends on the cut type \( c \); vertical boundary distances are used for horizontal cuts, and
Figure 5.8: The algorithm segments a design into rectangular regions with three criteria. First, segmentation boundaries, or cuts, should avoid intersecting elements. (a,b) show the intersection penalty for horizontal and vertical cuts; a cut placed near an element center would pay a high penalty. Second, regions should contain only text or graphics. Third, cuts should lie near the parent region’s center. (c) shows a final segmentation.

Second, the algorithm prefers that regions contain only text or graphical elements. An energy term evaluates the sub-regions $r_1$ and $r_2$ and counts the number of elements of the same class (text or graphics) in both regions.

$$F_{elm}(c) = -(N(r_1) + N(r_2))$$

where $N(r)$ defines the number of elements in region $r$ if all elements have the same class, and 0 otherwise.

Third, the algorithm prefers evenly splitting regions, so the normalized distance of the cut to the region center $r_c$ is:

$$F_{cen}(c) = \frac{|c - r_c|}{r_l}$$

where $r_c$ is the region center’s location, and $r_l$ is the region length. The system evaluates a cut using the following function:

$$F(c) = w_{int}F_{int}(c) + w_{elm}F_{elm}(c) + w_{cen}F_{cen}(c)$$
In all experiments, \( w_{int} = 100, w_{elm} = 100, w_{cen} = 1 \). Using the cumulative sum of the intersection distances allows the parse to be computed quite efficiently (5-10 ms for a 100 × 150 size image). Fig. 5.2 (right) and Fig. 5.8 shows two examples and visualizations of the intersection costs.

### 5.4 Graphic Design Energy Terms

We next describe our energy-based model of graphic design. Our model is designed to balance many goals, such as clearly conveying information, aesthetics, and stylistic variation. Our modelling choices are inspired by principles gleaned from the graphic design literature \([37, 165, 73, 166]\), and our own observations. We outline these principles, and the energy terms we created to capture them, as well as others we found necessary. As we describe later, to generate designs we will minimize these energy terms, therefore lower values of energy terms are preferred.

Building a model for graphic design is challenging for several reasons. We want an energy function that captures a range of styles. However, the model must also produce a consistent style on designs with different content. Furthermore, many principles of design, and their corresponding energy terms, are related and interact with one another. For example, white space is an important part of graphic design. But many aspects of design affect white space, such as element scale or symmetry. Finding a working set of energy terms is non-trivial; a variety of terms are required to capture a range of styles, but, with too many, the model may overfit while learning.

Lastly, modelling graphic design well is difficult because of our exposure to it and sensitivity to slight mistakes. Most people see many graphic designs daily, so we are attuned to successful designs. It is worth noting that, while previous work has proposed related terms, none has created a full model which can synthesize designs such as posters or advertisements in a variety of styles.

Our methodology involved studying design literature and examples for stylistic variation or functionality. We then created relevant energy terms, optimized designs with these terms, then refined the terms and re-optimized. Parameters were initially hand-tuned, but, as the model increased in complexity, Nonlinear Inverse Optimization (NIO) (Sec. 5.6) was used to learn model parameters. NIO was also used to remove redundant energy terms; if after learning on several styles, an energy term was not used, it was removed. For example, one term measured the fraction of the design covered by graphical elements, and was removed after it was found to be redundant. In Fig. 5.9, we show the effect of removing various energy terms from the model.
5.4. Graphic Design Energy Terms

Figure 5.9: Model terms. Given the two training layouts, the system attempts to learn the style with different energy terms removed.

5.4.1 Alignment

Correct alignment is an important aspect of good design; poor alignment is both distracting and confusing. The model prefers that elements align with each other. One energy term measures
the fraction of element pairs that align with a type $a$ such as Left or Top:

$$E^a_{\text{align}} = -S \left( \frac{1}{n^2} \sum_{i \in (\text{all})} N^a_i + \sum_{j \in (\text{all})} I^a_{ij} \right); \alpha_a$$  \hspace{1cm} (5.11)

where $n$ is the number of elements and $I^a_{ij}$ indicates if element $i$ and $j$ are aligned by type $a$. We define separate energy terms for different alignment types. $N^a_i$ indicates if $i$ is a multi-line text element internally aligned by type $a$. For example, the date in Fig. 5.11 has an X-Center internal alignment. Each feature is transformed by a sigmoid $S(x; \alpha_a)$. The sigmoid is a smooth-step function, with the parameter $\alpha_a$ controlling the smoothness of the step, and is used to shape energy terms. See Fig. 5.10.

The model penalizes misalignments with the following term:

$$E^x_{\text{misalign}} = \frac{1}{3n^2} \sum_a \sum_{i \in (\text{all})} \sum_{j \in (\text{all})} I^a_{ij} C(d^a_{ij})$$  \hspace{1cm} (5.12)

where $C(d^a_{ij})$ is a robust cost function for a given distance which heavily penalizes slight misalignments: $C(d) = 5 \arctan \left( \frac{d}{0.01} \right)$. $d$ is the distance in pixels normalized by the design size. $E^y_{\text{misalign}}$ is defined similarly for $y$-axis alignments.

The model also encourages larger alignment groups:

$$E^g_{\text{group}} = -S \left( \frac{1}{nm} \sum_g \sum_{i \in (\text{all})} I^g_i; \alpha_{ag} \right)$$  \hspace{1cm} (5.13)

where $n$ is the number of elements, $m$ is the number of alignment groups and $I^g_i$ indicates if element $i$ belongs in alignment group $g$. See Fig. 5.7 for examples.
An alternative to this approach would be to include the alignment detection when evaluating this energy term, as in Vollick et al. [161]. However, there are a few advantages to separating the analysis step into a separate module. This separation allows a simpler energy function. Alignment detection is also used in the optimization step to reduce misalignments.

5.4.2 Balance

We measure the global symmetry using a binary map of text or graphical elements, flipped along an axis. More precisely, for vertical or x-axis symmetry:

\[
S_{x\text{-symm}}^{\text{text}} = \sum_{c \in \{\text{text}\}} \left( \frac{\sum_p |I_p^c - I_{\text{flip}(p,x,G)}^c| I_p^c}{\sum_p I_p^c} - 1 \right)
\]

\[
E_{x\text{-symm}}^{\text{text}} = S(S_{x\text{-symm}}^{\text{text}}; \alpha_{txs})
\]

(5.14)

(5.15)

where \(I_p^c\) is a binary variable indicating if pixel \(p\) is of class \(c \in \{\text{graphic, text}\}\), and \(\text{flip}(p,x,G)\) indicates the symmetric counterpart of pixel \(p\) along the x-axis of image \(G\). We define a similar term \(E_{y\text{-symm}}^{\text{text}}\) with \(\text{flip}(p,y,G)\) indicating y-axis symmetry. We also define an asymmetry term \(E_{x\text{-asymm}}^{\text{text}} = S(S_{x\text{-symm}}^{\text{text}} - 1; \alpha_{txa})\), and identical terms for graphical elements, giving eight global symmetry terms total.

Designers often symmetrize elements not with the overall page, but with regions of the page (Fig. 5.11). Given the hierarchical segmentation of Sec. 5.3, we define \(E_{\text{regionSymm}}^{\text{text}}\) as above, using the symmetric counterpart of pixel \(p\) along the x-axis in region \(r\).

5.4.3 Emphasis

The model also matches the perceived importance of elements to their desired importance. Using the method of Sec. 5.3.1, the system first estimates the hidden variable \(q_i\), the perceived visual importance of element \(i\). Estimated values are compared to fixed scalar importance values \(F_P^i\) for each element using Pearson correlation:

\[
E_{\text{emp}}^{\text{text}} = -S(-\text{corr}(q, F_P^i); \alpha_{emp})
\]

(5.16)

where \(\text{corr}(x, y)\) is the Pearson correlation between paired samples \(x\) and \(y\). \(q\) and \(F_P^i\) are vectors of the perceived and desired importance values for all elements. \(E_{\text{emp}}^{\text{graphical}}\) is defined similarly for graphical elements.

The desired importance values are provided as metadata during the design creation process and are not estimated by the system. In practice, these values are usually simple to specify.
Figure 5.11: Symmetry types. Global symmetry is defined over the entire design; region symmetry is defined with respect to the segmentation regions (show with red lines). The top element has both symmetry types; the elements in the two bottom regions have only region symmetry.

White [165] recommends designers establish an importance hierarchy of at most 3 levels, as more can become confusing. In our examples we usually specify 3 levels of importance: low, medium, and high, though occasionally we used 4 levels.

5.4.4 White Space

The model encourages white space with the following term:

$$E_{\text{whiteSpace}} = -S \left( \frac{\sum_p I_p}{wh}; \alpha_{ws} \right)$$  \hspace{1cm} (5.17)

where $I_p$ is a binary variable indicating if pixel $p$ is not covered by an element, and $w$ and $h$ are the design width and height. The model penalizes large regions of empty white space using the cubed distance to an element over the entire image:

$$E_{\text{spread}} = S \left( \frac{1}{wh} \sum_p \min(D(p, M_i)^3); \alpha_{spr} \right)$$  \hspace{1cm} (5.18)

where the sum is over all pixels in the design $p$, and $D(p, M_i)$ is the Euclidean distance of pixel $p$ to element $M_i$. For efficiency, we compute the distance transform on an binary image of element placements. Pixels inside the element receive a distance of 0.

The model encourages separation between elements. The distance between elements $i$ and
\(j\) is denoted as \(\hat{d}_{ij}\), the minimum of the mean squared distance between elements:

\[
\hat{d}_{ij} = \min_{p \in I} \sqrt{\frac{1}{2} (D(p, M_i)^2 + D(p, M_j)^2)}
\]

To avoid computationally-expensive distance transforms during optimization, distance maps from the element boundary are pre-computed for an area around each element and scaled. If the maps for two elements do not overlap, the bounding box distance is used.

The distance energy is the mean of the nearest distance for each element:

\[
E_{\text{dist}} = \frac{1}{n} \sum_{i \in \text{all}} \left( 1 - S\left( \min_{j \in \text{all}} (\hat{d}_{ij}); \alpha_{\text{dist}} \right) \right)
\]

\(E_{\text{textDist}}\) is defined similarly for text elements.

The model encourages uniform vertical spacing of text elements:

\[
E_{\text{textSepVar}} = S(\text{var}(\cup_{i,j\in \text{text}} d_{ij}^y); \alpha_{\text{ism}})
\]

where \(\text{var}(x)\) is the variance of a set \(x\). \(\cup_{i,j\in \text{text}}\) indicates all pairs of text elements that overlap along the \(x\)-axis, that also have no other elements between them. \(d_{ij}^y\) is the vertical bounding box distance.

Border margins \(M_i^k\) for each element are the distances of the bounding box edge to the respective boundary. The energy is the mean of the nearest margin distances:

\[
E_{\text{textMargin}} = \frac{1}{n} \sum_{i \in \text{text}} \left( 1 - S\left( \min_k (M_i^k); \alpha_{\text{tmi}} \right) \right)
\]

\(E_{\text{margin}}\) is similarly defined for graphical elements.

### 5.4.5 Scale

Element scale is an important practical and stylistic decision. Elements must be large enough to view, but not so large that the design becomes cluttered and aesthetically displeasing. For a given layout, the size of a text element \(M_i^t\) is defined as the element height \(M_i^h\) divided by the number of lines \(F_i^l\), weighted by a scaling parameter \(\tau_s\), and normalized by the design height \(h\). That is, \(M_i^t = (\tau_s M_i^h)/(F_i^l h)\). For almost all examples in this chapter, \(\tau_s = 1\). Using a smaller value, for example \(\tau_s = 0.4\), will increase the element sizes. The size for graphical elements is the bounding box area \((M_w^i M_h^i)/(wh)\) where \(M_w^i\) and \(M_h^i\) are element's bounding box width and height, and \(w\) and \(h\) are the design width and height. We encourage larger text with the following term:

\[
E_{\text{textSize}} = -\frac{1}{n_t} \sum_{i \in \text{text}} S(M_i^t; \alpha_{ts})
\]
where \( n_t \) is the number of text elements; \( E_{\text{graphicSize}} \) is defined similarly.

Energy terms also penalize the variance of text and graphic sizes:

\[
E_{\text{textVar}} = S(\text{var}(\cup_{i \in \text{text}} M_i^t); \alpha_{tv})
\]

The model prefers elements to be above a minimum size:

\[
E_{\text{minTextSize}} = \sum_{i \in \text{text}} \max(\tau_t - M_i^t, 0)
\]

\( E_{\text{graphicVar}} \) and \( E_{\text{minGraphicSize}} \) are defined similarly. The minimum sizes for text and graphics were set to \( \tau_t = 0.0275 \) and \( \tau_g = 0.04 \).

### 5.4.6 Overlap and Boundaries

Overlapping elements are common in many designs. We define several types of overlap, including overlap of elements on text, overlap of text on graphical elements, and overlap of graphical elements on other graphical elements. Separate energy terms are defined for each overlap type, and are summed over overlapping pixels \( p \). In our model, we render all graphical elements before text, to prevent obscured text. This also prevents the draw order of elements from affecting the overlap.

\[
O_{\text{textOverlap}} = \frac{\sum_p A_p'}{wh}
\]

\[
E_{\text{textOverlap}} = S(O_{\text{textOverlap}}; \alpha_{to})
\]

where \( A_p' \) indicates the pixel’s alpha component of any element overlapping any text element. We define \( E_{\text{graphicTextOverlap}} \) and \( E_{\text{graphicGraphicOverlap}} \) similarly to indicate text overlapping graphical elements, and graphical elements overlapping each other.

For graphical elements, users may also provide fixed binary masks as metadata, to prevent overlapping important regions such as faces or logos. The model includes an energy term which measures the overlap in these regions:

\[
O_{\text{impOverlap}} = \frac{\sum_p F_p^{\text{no}} A_p^g}{wh}
\]

\[
E_{\text{impOverlap}} = S(O_{\text{impOverlap}}; \alpha_{no})
\]

where \( F_p^{\text{no}} \) indicates if the binary mask has been drawn on pixel \( p \), and \( A_p^g \) is the alpha component of any overlapping element.
Lastly, the model penalizes text overlapping graphical elements where a low contrast might make them difficult to read:

\[ E_{textContrast} = \frac{1}{n_t} \sum_{ic(text)} S(M_i^{con}, \alpha_{ic}) \]  

(5.30)

where \( M_i^{con} \) is a basic text contrast measure defined using the absolute difference of the RGB values before and after rendering for an element. The mean difference for each vertical line of the difference images are sorted, and the mean of the worst 20% used. This measure produces a high penalty even if a small part of the text has low contrast.

The model controls how much elements may extend past the boundaries of the design:

\[ B_{graphic} = \frac{1}{n} \sum_{i \in (graphic)} \left( 1 - \frac{\sum_{pe,i} A_p I_p}{\sum_{pe,i} A_p} \right) \]  

(5.31)

\[ E_{graphicBoundary} = S(B_{graphic}; \alpha_{tg}) \]  

(5.32)

where \( \sum_{pe,i} A_p \) denotes the sum of the alpha values for all pixels in element \( i \). \( \sum_{pe,i} I_p A_p \) denotes the sum of alpha values which are within the design boundaries. A similar energy term for text and important regions \( E_{impBoundary} \) is defined.

### 5.4.7 Flow

A good design layout presents information in a clear read order. However, modelling the visual flow of graphic designs is a difficult open problem. To address this issue, our model uses simple positioning heuristics. First, more important text elements are placed higher and to the left of less important elements:

\[ E_{flowX} = S \left( \sum_{i \in (all)} \sum_{j \in (all)} L_{ij} d_{ij}; \alpha_{flowX} \right) \]  

(5.33)

\[ L_{ij} = \begin{cases} \max(F_p^j - F_p^i, 0) & \text{if } M_i^l \leq M_j^l \\ \max(F_p^i - F_p^j, 0) & \text{otherwise} \end{cases} \]  

(5.34)

where \( M_i^l \) is the left boundary or center of element \( M_i \), depending on which is closest between the two elements. The difference in desired importance \( F_p^j - F_p^i \) is weighted by the two elements’ bounding box distance \( d_{ij} \). Similar terms are defined for the \( y \)-axis. See Fig. 5.12 for an example.

The overall location of graphical and text elements also affects the design style. The model uses a simple positioning heuristic using the mean of the element centers:

\[ E_{x-location}^{text} = -\frac{1}{n_t} \sum_{i \in (text)} \frac{M_i^x-center}{w} \]  

(5.35)
Figure 5.12: The model uses simple heuristics for specifying read order. Left: unimportant elements are above and to the right of more important elements. Right: corrected design.

We also define a reverse term by calculating the mean of $1 - \frac{M^x_{center}}{w}$, similar terms for the $y$-axis, as well as terms for graphical elements, giving 8 total.

We also measure the variance of the element positions for both text and graphical elements:

$$E_{\text{x-variance}}^{\text{text}} = S\left( \alpha_{x-var} \left( \var_{i \in \text{text}} \left( \frac{M^x_{center}}{w} \right) \right) \right)$$

(5.36)

5.4.8 Unity

Another important design principle is unity, when elements appear to belong together. This principle is closely related to Gestalt psychology’s Law of Proximity, i.e., elements which are near are visually grouped [164]. We model element unity by allowing users to group elements with scalar group IDs provided as metadata. The model encourages group members to have a similar size:

$$E_{\text{groupSizeVar}} = \frac{1}{|G|} \sum_{g \in G} \var(\cup_{i \in g} M^x_i)$$

(5.37)

where $|G|$ is the number of user groups, $\cup_{i \in g}$ indicates all the elements in a group $g$, and $M^x_i$ is the size of element $i$.

Group members are encouraged to have a similar perceived importance. $E_{\text{groupImpVar}}$ is defined as above using $q_i$. We enforce a weak position constraint that group members should be
close:

$$E_{\text{groupDistMean}} = \frac{1}{|G|} \sum_{g \in G} \frac{1}{n_g} \sum_{i \in g} \min(d_{ij})$$  \hspace{1cm} (5.38)

where $n_g$ is the number of elements in group $g$, and $d_{ij}$ is the bounding box distance between elements.

### 5.4.9 Previous Layout

When improving or retargeting designs, the model uses a previous layout of the same elements. We often wish to preserve properties of the design including relative locations, sizes, and importance. Given a previous design, the model includes the following terms:

$$E_{\text{prevHeight}}^\text{text} = S\left( \frac{1}{n} \sum_{i \in (text)} |M_i^h - M_i^{0h}|; \alpha_{ph} \right)$$  \hspace{1cm} (5.39)

$$E_{\text{prevPosition}}^\text{text} = S\left( \frac{1}{n} \sum_{i \in (text)} ||M_i^c - M_i^{0c}||_2; \alpha_{pp} \right)$$  \hspace{1cm} (5.40)

$$E_{\text{prevImp}}^\text{text} = S\left( \frac{1}{n} \sum_{i \in (text)} |q_i - q_i^0|; \alpha_{pi} \right)$$  \hspace{1cm} (5.41)

$$E_{\text{relHeight}}^\text{text} = S\left( \frac{1}{n^2} \sum_{i,j \in (text)} |r_{ij}^h - r_{ij}^{0h}|; \alpha_{pr} \right)$$  \hspace{1cm} (5.42)

where $M_i^h$ and $M_i^{0h}$ are the current and original heights of element $i$, $M_i^c$ and $M_i^{0c}$ are the relative positions of the element centers in the current and original design, and $q_i$ and $q_i^0$ are the current and original perceived importance values. $E_{\text{relHeight}}^\text{text}$ enforces relative differences in heights: $r_{ij}^h = M_i^h - M_j^h$. Similar terms are defined for graphical elements.

The model also compares two global properties of the designs, overlap and elements beyond the boundaries:

$$E_{\text{prevOverlap}} = S(|O^c_{gt} - O^o_{gt}|; \alpha_{po})$$  \hspace{1cm} (5.43)

$$E_{\text{prevGraphicBoundary}} = S(|B^c - B^o|; \alpha_{pg})$$  \hspace{1cm} (5.44)

where $O^c_{gt}$ and $O^o_{gt}$ measure text overlapped on graphical elements in the current and original designs, and $B^c_{\text{graphic}}$ and $B^o_{\text{graphic}}$ measure the fraction of graphical elements which extend beyond the boundary. See Sec. 5.4.6 for details on these properties.
The energy function is extremely multi-modal, and the variables are highly coupled because of alignment constraints. To optimize this difficult problem, we follow previous work in layout optimization and use simulated annealing [2, 168]. The optimization takes an initial layout where elements are placed along the left boundary and proposes changes to the elements’ positions and scales. Proposed changes resulting in lower energy are always accepted. Early in the optimization, there is a greater probability of accepting layouts with higher energy, allowing escape from local minima. As the optimization progresses, the algorithm is less likely to accept higher-energy layouts. See Appendix C for details of the proposal distribution.

Optimization takes approximately 40 minutes on a MacBook Pro (Intel Xeon 2.6 Ghz). The optimizer runs for 30,000 iterations, using a linearly-decreasing temperature. The highly multimodal nature of the problem often results in the optimizer finding different local minima. This can be useful for sampling layout suggestions for a designer however (e.g., [99]), as the local minima are often visually different. For each result in this chapter, we ran 8 optimizations in parallel and selected the minimum. Fig. 5.13 shows several examples of synthesized layouts.
5.6 Learning Model Parameters

The parameter vector $\theta$ that defines a layout energy includes 122 parameters. Manually setting this large number of parameters is prohibitively time-consuming. We use Nonlinear Inverse Optimization (NIO) [82, 161] which learns nonlinear model parameters $\theta$ based on one or more examples. NIO is an instance of structured learning [107], though unlike most structured learning methods, NIO works for nonlinear parameters and continuous outputs.

NIO learns parameters based on one or more examples. Given an example layout $X_T$, we assume it is optimal according to a parameter vector $\theta$ which we want to find. We express this with the following objective function:

$$G(\theta) = E(X_T; \theta) - \min_X E(X; \theta)$$

(5.45)

This function says that we want to minimize the difference in energy between the example layout $X_T$ provided by the designer, and the optimal layout for this $\theta$, using the given elements. If we find a global minimum at $G(\theta) = 0$, then we have found the $\theta$ that makes $X_T$ optimal.

Given the complexity of the model, we find it useful to add a weighted prior on the parameters:

$$G(\theta) = E(X_T; \theta) - \min_X E(X; \theta) + \lambda \sum s(\theta_i - \hat{\theta}_i)^2$$

(5.46)

where $s$ is a binary vector and $\hat{\theta}$ is the initialization of the parameter vector. For example, if the algorithm learns from a design with slight misalignments, this prior can still enforce that alignment errors should be penalized heavily. We use $\lambda = 10$ for the retargeting and improvement applications, and $\lambda = 0$ for the design styles application. To avoid negative parameters, we re-parameterize $\theta_i = \exp(\beta_i)$ and optimize for $\beta_i$ to ensure that $\theta_i > 0$.

To evaluate $G(\theta)$, we must first compute an optimal layout (the min term). We approximate this optimal layout using optimization as described in the previous section. We then minimize $G$ using gradient descent with line search. When $G$ is differentiable

$$\frac{dG}{d\theta} = \frac{\partial}{\partial \theta} E(X_T; \theta) - \frac{\partial}{\partial \theta} E(X_S(\theta); \theta) + 2\lambda s(\theta - \hat{\theta})$$

(5.47)

$$X_S(\theta) = \arg \min_X E(X; \theta)$$

(5.48)

since $\frac{\partial E}{\partial X} \frac{dX}{d\theta} = 0$ if $\frac{dX}{d\theta}$ exists [161]. Intuitively, following the gradient direction has the effect of reducing the energy of the training example $X_T$ while increasing the energy of the counter example $X_S$, as visualized in Figure 5.14. When learning with multiple examples, we decrease the mean energy difference over all training examples. See Algorithm 1 for pseudocode.
5. Learning Layouts for Single-Page Graphic Designs

Figure 5.14: Intuition for NIO. The goal is to find a $\theta$ for which $X_T$ is at the bottom of the energy function. Initially, $X_T$ is not at the bottom. In each step the algorithm generates a layout $X_S$ with lower energy than $X_T$, and then adjusts $\theta$ to push $X_T$ down and $X_S$ up. From Liu et al. [82].

NIO can be seen as a form of Contrastive Divergence learning [47]. Inspired by Persistent Contrastive Divergence [156], we persist all previous counter examples to improve performance and efficiency. At the start of each iteration, the training example and previous counterexamples are checked to find the lowest energy and this layout used to initialize the optimization. Every 5 iterations, we perform a longer optimization to escape local minima.

5.7 Applications

We demonstrate our model with three applications: design synthesis in different styles, design retargeting, and design improvement. Separate parameters are learned for each application using NIO and a small number of examples, as the energies of each application differ significantly. For learning styles, there is no previous layout and the system attempts to match a particular style. Both retargeting and improvement measure the difference from a previous layout, but the energy is different. For example, when retargeting it is more important to match the previous layout’s perceived element importance.

Design Synthesis. The most basic goal of the system is to generate design layouts in a variety of styles. Fig. 5.13 shows examples of synthesized layouts. Based on two example layouts, the system learns the style parameters with NIO and then generates other layouts in the same style. Learning using a single layout is possible, but, because of the model complexity, we found using two similar layouts reduced over-fitting and produced better results.

In Fig. 5.15 we demonstrate our approach with three landscape-ratio styles: a simple sym-
Algorithm 1 Pseudo-code for Nonlinear Inverse Optimization (NIO).

```plaintext
function NONLINEARINVERSEOPT(X_T, λ, ̂θ, s)

    ̂θ ← ˆθ
    i ← 1
    C ← X_T

    while not done do
        if i mod 5 = 1 then
            Perform long optimization
            X_S ← Optimize( ̂θ, [])
        end if
        else
            Find lowest energy layout X_I to start optimization
            X_I ← argmin_{X ∈ C} E(X; ̂θ)
            X_S ← Optimize( ̂θ, X_I)
        end if

        Add new counter example X_S to working set C
        C ← C ∪ X_S
        Δθ ← \frac{∂}{∂θ} E(X_T; θ) − \frac{∂}{∂θ} E(X_S; θ) + 2λs( ̂θ − ˆθ)
        ρ ← LineSearch( ̂θ, Δθ, X_T)
        ̂θ ← ̂θ − ρΔθ
        i ← i + 1
    end while

end function
```

Design Retargeting. The system can retarget designs to new sizes. Retargeting is an important task for designers as designs are now often viewed in a variety of sizes and aspect ratios. In our tests, we primarily focus on retargeting between landscape and portrait sizes, but our approach works for arbitrary sizes (Fig. 5.17). Model parameters were learned from 12 pairs of designs in two different aspect ratios (landscape and portrait). For each design in the pair, the alternate design was used in the previous layout energy terms of Sec. 5.4.9, giving 24 examples.
Figure 5.15: Learning design styles. Parameters are learned from the top two examples of each column and used to generate layouts for other designs. (a) a simple symmetric style, (b) two columns with center-aligned elements (c) a large center graphic with smaller text surrounding.
Figure 5.16: **Learning design styles.** Style parameters are learned from the left two examples and used to generate layouts for other designs. (a) highly symmetric style with smaller elements and large margins, (b) asymmetric style with larger graphics and text, (c) higher placed graphics with larger, left aligned text.

total. Retargets on new designs were generated using these learned parameters.

To evaluate our algorithm, we hired a professional graphic designer who was experienced in resizing posters and other print materials to different dimensions. The designer was provided 98 original designs and created layouts in a new size (from portrait to landscape or vice-versa) which matched the original design's style while also being aesthetically pleasing. The designer faced the same constraints as our system, and could only translate and scale elements. In Fig. 5.18
we show several retargeting examples. We then compared our automatic retargets using A/B comparisons on Mechanical Turk (MTurk). 45 users were paid 5¢ to compare 10 designs based on aesthetics and similarity to the original design. Duplicates were added in each task and inconsistent workers removed.

To evaluate how well our algorithm compares to novice users, we re-ran the previous designer retargeting experiment using MTurk. Users used the same custom retargeting software that was provided to the designer. Users were paid 5¢ per retarget, and took a median of 2.5 minutes per design, similar to the 2.4 minutes for the professional designer. Because of the subjectivity of the task, to encourage higher quality results, users were also informed that designs would be evaluated by other workers and bonuses paid for the best retargets. In a second study, users were shown 9 MTurk retargets and our automatic retarget in random order, and ranked the retargets from best to worst. Workers were paid 5¢ per evaluation. 25 workers were used, with the most inconsistent of those users removed. The top 20% of designs received 15¢; the top 10% of rankings received 25¢. To measure a given user’s design ranking, we computed the difference between the user’s ranking, and the ranking averaged over all users. Fig. 5.20 shows
Figure 5.18: Portrait-to-landscape retargeting. We show retargeting results from an example layout by a professional designer, the best crowdsourced retarget (out of 9), and our automatic retarget. The designer, MTurk users, and the algorithm could all modify element position, scale, and text alignment, but not line-breaks or rotation.

an example of the top 5 retargets with their average rankings.
retargeting. In Fig. 5.18 (c) the automatic retarget has produced a reasonable layout, though fairly different than human retargets.

The mean preference of our automatic retargets compared to designer retargets, after removing any layouts used as training data, was 0.39 (see Fig. 5.21). However, analyzing the individual
Figure 5.20: Retarget ordering. The portrait retargets were manually created by MTurk users in one task, and ranked in a second task. The ‘Automatic’ retarget was created by our system. The mean/std dev of all rankings are reported, with the designs sorted by mean rank.

Figure 5.21: Automatic retarget evaluation. Left: mean preference of A/B comparisons between automatic and designer retargets. Middle: histogram of tests which had a statistically significant preference for either retarget. Right: histogram of rankings for automatic retargets compared to novice MTurk users (1 is the best). Our automatic retargets cannot beat designer retargets, though often perform comparably, and are often highly rated compared to novice humans.
A/B tests using the binomial test show that many failed to show a statistically significant preference for the designer retargets, suggesting the algorithm often retargets reasonably. To compare with novice MTurk retargets, we performed the ordering test described earlier. Our retargets achieved an average rank of 4.61 (std. err of 0.29). Fig. 5.21 shows a histogram of the rankings for our automatic retargets after sorting by the mean rank. While our approach cannot consistently beat the best human retargets, we do perform better than the average MTurk user. We use a one-sample Student-\(t\) test to evaluate if the mean of all automatic retarget rankings is less than 5.5, the mid-point between ranks 1 and 10 (\(p < 0.05\)). As further support for our model, in Sec. 5.7.1 we show statistically significant correlations between many energy terms and the ranking scores.

Considering the problem’s difficulty and lack of previous automatic algorithms, merely replicating novice human ability is significant. Furthermore, while users usually prefer designer retargets to ours, they often make no distinction between the two, indicating that our automatic retargeting is often doing as well as a professional.

**Design Improvement.** The system can take an existing layout and improve it to better match principles of graphic design. Parameters are learned using 12 examples of an original and improved design. Given a new design, the learned parameters are used to optimize a new improved version. Fig. 5.22 shows the results of our approach improving a variety of designs, from very poor initial layouts which are changed significantly, to good initial layouts which are only changed slightly. For example, Fig. 5.22(a) is improved by grouping and aligning the text elements. Fig. 5.22(d) is improved by increasing the graphic size while aligning the elements. Fig. 5.22(b) and (e) are changed more significantly, by changing the placement of elements to improve the read order, as well as improving alignment, and symmetry. The right columns gives some failure cases of our algorithm. In Fig. 5.22(c) the symmetry of the text in the original is lost, producing a worse design. In Fig. 5.22(f), the new design is significantly worse due to a poor grouping of elements in the top, resulting in smaller text and a more confusing read order.

To evaluate our improvement approach, we performed another MTurk study. The manually created and ordered designs from the retargeting task were divided into three groups: the best rated designs, the worst rated designs, and all designs. Out of the 9 human retargets, the top 2, bottom 2, and 2 random designs were each chosen, giving 196 designs per set. These designs were then optimized with the learned improvement parameters. Finally, 45 MTurk users selected their preferred design in an A/B comparison, with a randomized left/right position for the improved design. Users were paid 5¢ to compare 10 designs based on aesthetics and clarity. Duplicates were added in each task and inconsistent workers removed.
Figure 5.22: Design improvement. Based on layouts created by MTurk users (top design), we generate improved layouts (bottom design). $p_i$ are the fraction of MTurk users who prefer the improved designs. The right column shows failure cases.

Fig. 5.23 shows the overall preference for the original designs compared to the improved versions. We also show a histogram of the tests which had a statistically significant preference for the original or improved (using the binomial test with $p < 0.05$), or where there was no statistical difference. Our approach often improves the worst designs while matching the best designs. For the overall set of designs, our approach often improves the design or produces
no difference; only rarely does the algorithm produce a worse design. These results show that the system successfully models basic principles of graphic design, and can generate appealing layouts.

5.7.1 Energy Terms and Score Correlation

We can further analyze our model using the retargeting designs from MTurk users. Each of these 880 designs has an associated set of energy term values, as well as a ranking score. By computing the Pearson correlation between individual energy terms and the scores, we can determine which terms are the most closely tied to the quality of the designs. Note that these scores are not ideal, as they are relative to other designs. To partially alleviate this issue, for each term we subtract the mean over all 10 designs in the ranking set. To evaluate statistical significance, we use a significance level of 0.05/70 = 0.0007; the Bonferroni correction on the significance level is required, as we are comparing 70 terms.

As Table 5.3 shows, there are many statistically significant, though fairly weak, correlations with different energy terms. As expected, since this a retargeting test, the terms measuring the difference from the original layout all have a positive correlation. The highest correlation ($r = 0.39$) is for the difference in text position from the original layout. Some other high-level conclusions are that higher scores are correlated with bigger text ($r = 0.32$), less overlap of text on images is preferred ($r = 0.2$), elements should be spread out on the page ($r = 0.26$) (i.e., less
Table 5.3: Energy terms which show a statistically significant correlation with scores from MTurk users. The correlations were computed using features for 880 designs with the average scores given to those designs by other MTurk users.

white space), the model’s flow heuristic is useful ($r = 0.3$), and text should be lower on the page ($r = 0.25$).
5.8 Discussion

Automatic tools for understanding and creating graphic designs are important for both professional and novice designers. Design is an extremely difficult task, and the vast number of devices and viewing conditions for designs have increased the burden on designers significantly. However, many books on graphic design principles are vague and difficult to build tools from directly. By contrast, we model design principles explicitly, synthesize layouts using optimization, and directly evaluate modelling choices with user studies. This general approach allows a deeper understanding of graphic design principles, and will hopefully lead to tools for aiding novice and expert designers.

There are currently several limitations with our approach. Though complex, our model barely scratches the surface of possible graphic design layout styles. The model only optimizes element position and scale, and ignores rotations, font types, text line breaks, and optional elements. Other possible extensions include modelling of element read order and more complex constraints as in article layouts.

Our optimization and learning procedure described here are too slow for real-time interaction. Predicting element importance is currently an expensive operation, and also performs image-based operations like compositing. While we run multiple optimization in parallel, our approach is not intrinsically parallelizable due to the simulated annealing algorithm. However, parallel tempering has been used to parallelize layout synthesis on the GPU [99]. Inspired by this work, we next present a GPU-based model which greatly improves efficiency and allows our approach to be extended to an interactive design system.
6

Graphic Design with Automatic Layout Suggestions

The entire arrangement of my picture is expressive: the place occupied by the figures, the empty spaces around them, the proportions, everything has its share. Composition is the art of arranging in a decorative manner the diverse elements at the painter’s command to express his feelings.

– Henri Matisse, Notes of a Painter

Given the ubiquity of graphic designs, it is unsurprising that there are a multitude of design interfaces, from simple template-based tools like Powerpoint, to complex systems like Adobe Illustrator. Unfortunately, template-based tools are often restrictive, with limited assistance once the user deviates from a template. Complex tools permit a great deal of control, but are difficult to learn, provide little automatic assistance, and do not allow easy exploration of alternatives.

In this chapter, we present a novel system for graphic design using layout suggestions, and investigate different techniques for interacting with these suggestions. Our system proposes two high-level types of suggestions to the user, refinements which improve the current layout, and brainstorming suggestions which explore alternative layouts with large changes in style. See Fig. 6.1 for a screenshot. The system also permits retargetting of layouts, and allows users to easily create constraints on the suggested layouts. For example, the user can constrain groups of elements to be the same size.

A version of this chapter was published in the Proceedings of the ACM Conference on Human Factors in Computing Systems (Proc. CHI), 2015. [112]. This paper was a collaboration between the author, Aseem Agarwala, and Aaron Hertzmann. The project page for this chapter at http://www.dgp.toronto.edu/~donovan/design/, and contains a Supplemental Video which demonstrates the interface.
We use the energy-based design model introduced in Chapter 5 to propose suggestions. However, we adapt and simplify the model for use on the GPU, and use parallel tempering [147] to efficiently optimize layouts. Whereas a layout using the previous approach took approximately 40 minutes, all the layouts presented in this chapter took less than 10 seconds to create. To enable the brainstorming suggestions, we also learn a style space from example layouts. We first use Nonlinear Inverse Optimization (NIO) [82] to estimate parameters from example layouts, then principal component analysis (PCA) [123] to project to a low-dimensional subspace. This style space can then be sampled to generate new layouts in a variety of styles.

We also investigate two modes of interaction. First, we examine a suggestive mode, where suggestions are shown on the side and must be accepted. Second, we propose an adaptive mode where elements are moved automatically to improve the layout. The two modes are compared to a baseline without suggestions by novice users on Mechanical Turk, and the quality of the resulting layouts is also evaluated. We find that both modes produce significantly better designs than the baseline on average. Finally, we demonstrate the system’s use in tablet-based design.

Continuing our earlier parallel, this chapter is combines objectivist and subjectivist aesthetics, and is inspired by more modern aesthetic theories, such as those of Reber et al. [127], which emphasize the interaction between objective properties and the individual. While we use an objectivist energy function, the system’s end goal is a subjective one: a user’s exploration of the possible design space. Finally, learning individualized parameters on the energy function is an exciting area of possible future work, as weights could easily be updated given user interactions.

6.1 User Experience

In this section, we provide a high-level view of the design interface, shown in Fig. 6.1. See the Supplemental Video on the project site for a demonstration. A key goal of our interface is to provide assistance to a user during the design process. Specifically, we use an energy-based model to make small improvements to the existing layout, suggesting changes in position, scale, alignment, and line breaks. For example, if two elements are slightly misaligned, the model can suggest aligning them.

**Suggestive Mode.** In this mode, our interface shows three refinement suggestions on the left-hand side of the interface, which vary in their closeness to the current layout. The top layout is the most conservative, only making slight modifications, while the bottom layout will suggest larger changes to the layout. These suggestions are easy to view and accept; the user mouses
6.1. User Experience

Figure 6.1: Design interface. The central canvas allows the user to create layouts in a simple editor. On the left hand side, the system provides refinement suggestions, layouts which are similar to the current design, but are slightly improved. The refinement suggestions are updated whenever the user changes the central canvas. On the right hand side, the system provides brainstorming suggestions, large-scale changes in the layout in a variety of styles. These layouts appear in a scrolling list, unaffected by the central canvas.

over the suggestion in the left to see a full-size preview in the main canvas, then clicks to accept. These suggestions also adapt to intuitive user constraints. For example, if the user center-aligns two multi-line text blocks in the canvas, the interface will suggest internally center-aligning the text blocks. Fig. 6.2 shows an example of a canvas layout and the refinement suggestion.

Adaptive Mode. Our interface also works in a separate adaptive mode where elements are changed automatically. This mode provides a more fluid interaction, as the user does not need to view and accept changes. However, the adaptive mode is potentially frustrating for users if the model's suggestions do not match the user's desired goals. In Section 6.5 we investigate and evaluate both interaction modes, including user preferences and the quality of the resulting layouts.

Brainstorming Suggestions. Another important goal of the interface is to provide example layouts in a variety of styles. On the right hand side of the interface, we therefore provide a scrolling list of layouts. Each layout is distinct from the others, and the styles vary according to
Figure 6.2: Left: User interactions. Within the canvas, the user can move and scale elements, as well as changing the alignment and number of lines. Line breaks are controlled using the red diamonds; dragging outwards or inwards produces text blocks with fewer or more line breaks, respectively. The system supports snapping alignment lines; these lines are also used as temporary constraints in the suggestions. If the user does not want an element to change in the suggestions, they can lock elements by pressing the lock icon on the top right. Right: Refinement suggestion. The system has suggested left-aligning the elements, and increasing the size of the bottom graphic.

symmetry, text and graphical size, alignment preferences, etc. Figure 6.1 shows a few examples. In Section 6.4.2 we describe our sampling procedure in detail, and provide more examples. These brainstorming suggestions appear in both the suggestive and adaptive modes.

Retargeting. The system also allows simple retargeting of layouts to any desired size. If the user modifies the canvas size, the system will move the elements to match the previous relative locations and scales, while respecting design principles such as alignment or spacing between elements.

6.2 User Interactions

We next outline the user interactions of our system, and how the interactions are used to guide the model suggestions.

Basic Interactions. A user can move and scale the elements, as well as changing the alignment and number of lines, which reflows the text. Groups of elements can also be selected and moved together.
6.2. **User Interactions**

![Edit pane](image)

**Figure 6.3: Edit pane.** The user can modify the text, font, color, alignment, as well as line breaks and metadata used by the suggestions, such as the element importance and grouping.

**Accepting Suggestions.** A major part of our system is the proposal of different layouts. Users view the proposed layouts to the left and right of the canvas. Mousing over the proposal highlights the suggestion and previews it in the main canvas. The user may then click to accept. Users can also **Undo** and **Redo** any changes, either user modifications to the layout, or accepted suggestions.

**Locking.** The user can lock elements, which fixes the position and scale of the elements in the suggestions. If elements are unlocked, the system will suggest changes to the element position, scale, alignment, or number of lines. However, the suggestions use the canvas layout as a loose constraint, and will try to remain similar to the canvas layout. In Fig. 6.2, the user has locked the top-right graphic, so this element is fixed in the suggestion on the right.

**Alignment.** Correct alignment is an important part of visually appealing layouts. Our interface provides smart guidelines when aligning or resizing, which appear when two elements are close to aligned, and will snap elements together. These alignment lines are also used for suggestions; if an alignment line is created by the user aligning two elements in the main canvas, these elements will also be aligned in the suggestions. Fig. 6.2 shows an example where the user has left aligned the two text blocks. In the suggestion, the system has maintained this alignment, while also aligning the bottom graphic and changing the internal alignment of the top text block.

**Size Constraints.** The user can also specify size constraints on sets of elements, which will force these elements to have the same size. When one element is scaled in the canvas, the other elements are similarly scaled. These constraints are also respected in the refinement suggestions.
Saving Layouts. Our interface allows layouts to be saved for review later. Saved layouts are shown to the right of the main canvas, in a similar manner to brainstorming suggestions.

Modifying the Designs. Along with adding or removing elements, users can also modify any element in the design. In Fig. 6.3 we show the editing pane. For text blocks, users can also modify the font, alignment, line breaks, etc. For all elements, the user can modify metadata, such as the element importance. This metadata is used by the interface to make suggestions. For example, elements with higher importance are generally larger than less important elements. Once the design is modified, the system reinitializes the suggestions with the new design elements.

6.3 Design Model

Our system is designed to generate suggestion layouts, denoted by \( X \). These suggestions are usually constrained by the canvas layout \( X_c \), the central part of the interface where the user moves and scales elements. However, for brainstorming suggestions, the current canvas layout plays no role in creating the suggestion. The main goal of our graphic design model is an extremely fast evaluation of layouts, while also providing stylistic variation. We first describe the energy function used to model layouts, along with the user-provided constraints required for interactive design. We defer details of the optimization to Section 6.4.

6.3.1 Energy Function

As in Chapter 5, we measure the overall quality of a layout as a weighted sum of energy terms:

\[
E_h(X; \theta) = \sum_i w_i E_i(X, \alpha_i, X_c, C)
\]  

(6.1)

The suggestion layout \( X \) is defined as the \( x \) and \( y \) positions, height, alternate ID of each element. Alternate IDs select between different alternate elements, usually text blocks with different numbers of lines or different internal alignments. \( \theta \) are the model parameters, and are divided into two groups, \( \theta = [w, \alpha] \). Each energy term \( E_i \) has a positive weight \( w_i \), and a nonlinearity parameter \( \alpha_i \). The system may also use the canvas layout \( X_c \), and a constraint set \( C = (l, s, k) \). The constraint set includes \( l \), a set of alignment lines with associated elements, \( s \), a set of equality constraints for element sizes, and \( k \), a set of elements which are locked and must match the canvas layout.
The inputs to the model include the design elements (represented as bounding boxes and fixed metadata), parameters $\theta$, an output width and height, the canvas layout $X_c$, and constraint set $C$. Given the inputs and the model, our system optimizes $X$ to synthesize the suggestion layout. By changing the parameters $\theta$, we can generate layouts in various styles.

The metadata for each element include the type (graphic or text), aspect ratio of bounding box, importance value (scalar from 1-5), number of lines for text elements, group ID, and list of alternates. Alternates are other version of the elements with different number of lines or alignments. Including alternates for images, such as related images or different crops, is simple with our framework, but left for future work.

### 6.3.2 User Constraints

Within the canvas of the interface, the user can specify a set of constraints on the suggestions, both implicitly as the user positions elements in the canvas, and explicitly by locking elements and specifying size constraints.

**Canvas Layout Constraints.** The most important constraint on the suggestions is the canvas layout $X_c$, as the interface proposes refinements of this layout, not large changes. Elements must remain near their current position or scale on the canvas. By locking elements, a user can provide a hard constraint which fixes an element’s position and scale in the suggestions.

**Alignment Constraints.** When positioning elements, alignment lines appear to guide the placement of elements, and will also snap elements automatically in the canvas. These alignment lines then act as temporary constraints, with the elements in the suggested layouts matching the specified alignment. The internal alignment of multi-line text blocks will also change to match these alignment lines. Fig. 6.2 shows an example where the user has left-aligned two elements, which forms a constraint on the layout suggestion.

**Scale Constraints.** The user may also specify that a group of elements should have the same size, i.e., pixel height for graphics, and line height for text blocks. If any element in the group is scaled, the elements will all scale accordingly, preserving the aspect ratio. This constraint is also enforced in the suggested layouts.
6.3.3 Model Details

The interactive design model described next is an extension of the model of the previous chapter. However, the goal of rapid function evaluation on the GPU necessitates several changes. The interactive model also includes several new terms, such as those related to user constraints. Given the similarity to the previous model, we omit a full description of all the energy terms, and focus on important changes.

**Vector-based Elements.** Unlike the image-based elements of the previous model, elements are now simple bounding boxes. All previous energy terms which use element distances, overlap, and boundaries are thus much more efficient to compute.

**Constraint Terms.** The canvas layout, alignment lines, and size constraints all affect the suggestion layout. Specifically, there are energy terms to keep the suggestion layout near the canvas layout:

\[
E_{\text{matchHeight}} = S \left( \frac{1}{n} \sum_{i \in \text{all}} |M_i^h - M_i^{ch}|; \alpha_{ph} \right)
\]

\[
E_{\text{matchPosition}} = S \left( \frac{1}{n} \sum_{i \in \text{all}} \|M_i^p - M_i^{cp}\|_2; \alpha_{pp} \right)
\]

where \(M_i^h\) and \(M_i^{ch}\) are the suggested and canvas heights of element \(i\), \(M_i^p\) and \(M_i^{cp}\) are the element center positions. \(n\) is the number of design elements. \(S\) is a sigmoid \(S(x; \alpha) = \arctan(x\alpha)/\arctan(\alpha)\), and used to reshape energy terms, as large values of \(\alpha\) make the energy more sensitive to small changes when \(x\) is near zero.

There is also an energy term to match the alignment lines specified by the user. All pairs of elements associated with an alignment line are compared and the sum of distances between them penalized. There is also a penalty for having multi-line text elements with an internal alignment that does not match the alignment line:

\[
E_{\text{alignment}} = S \left( \sum_{l \in \text{all}} \sum_{(i,j,a) \in (l)} d_{ij}^a + N_i^a + N_j^a; \alpha_{al} \right)
\]

where \(l\) is an alignment line and \((i, j, a)\) are all associated elements \(i\) and \(j\) and the alignment type \(a\). \(d_{ij}^a\) is the distance between elements \(i\) and \(j\) for that alignment type. For example, if \(a\) has type Left, then \(d_{ij}^a\) measures the distance between the left boundaries of the two elements. \(N_i^a\) is a variable which indicates if \(i\) is a multi-line text element internally aligned by type \(a\).
In the interface, the user can constrain a set of elements to have the same size. There is therefore a term which penalizes any size difference:

\[ E_{\text{sizeConstraint}} = S \left( \sum_{s \in \mathcal{C}} \sum_{(i,j) \in (s)} \frac{|M^s_i - M^s_j|}{M^s_i} \right) \alpha_{\text{sc}} \]  (6.5)

Where \( s \) is a constraint on element sizes, \( i \) and \( j \) are all associated elements, and \( M^s_i \) is the size of element \( i \).

**Alignment.** Alignment is simplified in the interactive model, with alignment detected between all pairs of elements, and no grouping into larger sets.

**Symmetry.** In the previous model, a hierarchical segmentation was used to determine region-based symmetry. Region symmetry still exists in the interactive model, though in a simpler form. Element symmetry is now measured with respect to adjacent elements or the boundary:

\[ S^\text{ext}_{\text{regionSymm}} = \frac{1}{n} \sum_{i \in (\text{text})} \left( \frac{|M^l_i - M^r_i|}{M^l_i + M^r_i} \right) - 1 \]  (6.6)

\[ E^\text{ext}_{\text{regionSymm}} = S(S^\text{ext}_{\text{regionSymm}}; \alpha_{\text{txs}}) \]  (6.7)

Where \( M^l_i \) and \( M^r_i \) are the left and right distances for element \( M_i \) to the nearest element, or boundary if none is present, and \( n \) is the number of text elements. A similar term is defined for graphical elements.

**Importance.** The previous model used a time-consuming estimation of element importance. In the interactive model, element importance is now simply the size of the element. That is, bigger elements are considered to be more important.

**Diagonal Flow.** In the previous model, we used a simple heuristic that places less important elements below, and to the right of, more important elements. However, this heuristic is restrictive, so we modify the approach to allow less important elements to be placed to the left of more important elements, if they are below:

\[ E_{\text{flowDiag}} = S \left( \sum_{i \in (\text{all})} \sum_{j \in (\text{all})} L_{ij} d_{ij}; \alpha_{\text{flowDiag}} \right) \]  (6.8)

\[ L_{ij} = \begin{cases} \max(F^p_j - F^p_i, 0) & \text{if } M^x_i \leq M^x_j \land M^y_i \leq M^y_j \\ \max(F^p_i - F^p_j, 0) & \text{if } M^x_j \leq M^x_i \land M^y_j \leq M^y_i \\ 0 & \text{otherwise} \end{cases} \]  (6.9)
where $M_i^T$ is the top boundary of element $M_i$, and $M_i^L$ is the left boundary or center of element $M_i$, depending on which is closest between the two elements. The difference in desired importance $F_j^p - F_i^p$ is weighted by the elements’ distance $d_{ij}$.

**Spread.** In the previous model, we measured the spread of elements by computing the distance from each pixel to a design element. In this model, we compute the minimum distance to any element only for the corners and canvas center, which still encourages elements to spread out:

$$
E_{spread} = S \left( \frac{1}{5} \sum_p \min_i(D(p, M_i)^2); \alpha_{spr} \right)
$$

where $D(p, M_i)$ is the Euclidean distance of element $M_i$ to a point $p$. The point set includes the 4 corners and the center.

**Variable Line Breaks.** As part of the metadata for text elements, the user can specify a series of alternate elements with different line breaks and aspect ratios. The optimizer can then select between these different alternates. The model penalizes longer lines using the following term:

$$
E_{length} = S \left( \frac{1}{n} \sum_{i\in\text{\textit{text}}} (M_i^{ar})^2; \alpha_{len} \right)
$$

where $M_i^{ar}$ is the aspect ratio of element $M_i$.

### 6.4 Synthesizing Layouts

In this section, we describe our optimization strategy for synthesizing design suggestions at interactive rates, along with the sampling approach to create the brainstorming suggestions.

#### 6.4.1 Optimization

To optimize the energy function, we use parallel tempering [147], implemented on a GPU. Parallel tempering (PT) is a Markov Chain Monte Carlo (MCMC)-based optimization algorithm which runs $n$ independent MCMC samplers at different temperatures. States are swapped between the samplers at regular intervals based on the Metropolis criterion, allowing the ensemble to sample both high and low energy states. In Appendix D, we describe the sampler proposals.
Figure 6.4: We compare the performance of the standard parallel tempering algorithm with our two-stage approach. The plots are the mean function evaluations for 250 optimizations (50 runs for 5 different designs).

We also extend the standard PT algorithm by including a second-stage refinement optimizer, which optimizes the current best layout from the first-stage PT optimizer. When a new lower-energy state is found in the first-stage PT ladder, it sets all the states in the second-stage optimization to this new minimum. This approach allows the second-stage optimizer to explore the area around the current best solution. Fig. 6.4 shows the average performance of the optimizer, with and without this two stage refinement. The figure shows the mean optimizer value for 1000 iterations, averaged over 50 runs of 5 designs with varying numbers of elements (250 runs total). For a fair comparison, both optimizers have the same total number of samplers \( n \); in the two-stage model, there are two PT ladders of size \( n/2 \).

Optimizing a layout with no user constraints, as in the brainstorming suggestions, takes an average of 5 seconds. However, convergence for the refinement suggestions is much faster, as the canvas layout is used for initialization. This result is expected, as the refinement suggestions are constrained to be near the canvas layout. The optimizer usually converges to a reasonable solution less than a second, and is used to update the refinement suggestions on the left-hand side in the suggestive mode, or automatically shift the elements in the adaptive mode. Unfortunately, even 5 seconds is too long to wait for a brainstorming suggestion. The system therefore saves these suggestions, and if pre-computed layouts are available, will display them every 2 seconds. Such pre-computation is reasonable for our testing purposes. These brainstorming suggestions
could easily be parallelized on different GPUs. Furthermore, increasing the efficiency of the optimizer from 5 to 2 seconds could likely be accomplished with a faster GPU alone.

Note that the PT algorithm does not terminate, so a better solution can still be found after the layout has updated. However, this improved solution will be ignored. During interface prototyping, we experimented with allowing more than one update, but users found it confusing, particularly in the adaptive mode, since it was unclear when the automatic suggestions had converged. The current interface therefore waits until the user manipulates an element, and then produces a single update. This interaction is more fluid and intuitive: the user makes a change to the layout, then the system responds with a suggestion, and the process repeats.

6.4.2 Style Sampling

The brainstorming suggestions provide layouts in a variety of styles and arrangements. To generate different styles, we require different parameters for the model. Unfortunately, the parameterization of the energy function of Section 6.3 is high-dimensional, including 77 energy terms, each with a positive weight \( w_i \) and nonlinear parameter \( \alpha_i \). As described in the previous chapter, we use NIO to estimate parameters given an example, but we still require a method to generalize to new styles.

To accomplish this goal, we construct a low-dimensional subspace of the parameter space using basis vectors learned using PCA. Parameters are then sampled from this low-dimensional space. The algorithm is:

1. Given a set of training layouts \( S \) of size \( n \), run NIO on each layout \( l \in S \), and learn parameter \( p_l \).

2. Run PCA on the learned parameter matrix \( P \), and construct a new basis \( B \) of dimension \( n \), given the eigenvectors.

3. Sample randomly in the new basis \( B \), scaled by the square root of the corresponding eigenvalues.

In Fig. 6.5, we show several of the input layouts used to learn the subspace, as well as examples of the layouts created by sampling the subspace. In Fig. 6.6, we show that the learned design space also generalizes to different aspect ratios.
Figure 6.5: Learning a style space. Top: NIO is used to learn parameters for 19 designs, including the 6 example above. Given these examples, the system learns a low-dimensional subspace of the high-dimensional parameter space. Bottom: Given the subspace, we sample parameters to generate layouts in a variety of styles.
6.5 Evaluation

In this section, we evaluate our new interfaces against a baseline design interface which includes no suggestions, either refinement or brainstorming. Note that the alignment lines, snapping of elements, and resizing constraints remain even in the baseline. We evaluate two new interfaces against the baseline: the suggestive mode where the refinements are suggested on the left-hand side of the canvas, and the adaptive mode where elements update automatically. In both modes, the brainstorming suggestions are still shown on the right. We evaluate the interfaces in two ways. First, we ask users to create layouts in both interfaces and evaluate them with A/B testing to see if designs created with our new interfaces are better than designs created with the baseline interface. Second, we ask users to use both the baseline and a new interface, and then rate each interface.

6.5.1 Design Quality Study

In the first study, we used workers on Mechanical Turk to create layouts with three interfaces: adaptive, suggestive, and baseline. For all studies in this chapter, we required workers to be based in the USA, to help produce a more consistent set of users. Workers were paid $0.75 to create 3 layouts for a given design, using a single interface. We used 2 designs, with 20 workers each,
and randomly selected 60 layouts from each of the three interfaces, giving 180 designs total (3 interfaces × 30 layouts × 2 designs). The time taken with each interface was very similar. Users took an average of 2.41 seconds to create a layout in the baseline interface, versus 2.43 and 2.47 for the suggestive and adaptive interfaces respectively.

To compare the layouts, we used A/B testing, again on Mechanical Turk. For each new interface, we compared the 60 layouts with 4 random baseline layouts, and vice-versa for the baseline layouts. In the comparison task, 10 workers were paid $0.25 to compare 20 pairs and choose their preferred layout. Duplicates were added and inconsistent users removed.

Figure 6.7 (left) shows a histogram which counts the number of comparisons won by each interface, as well as ties. A layout ‘wins’ if it beats the alternate by a margin of 2 or more votes, out of the 10 total. Both new interfaces help novices create better layouts than the baseline interfaces, though the adaptive interface produces better designs overall, likely because we force model suggestions. These suggestions fix many basic layout problems for novices including overlap or poor alignment. Our suggestive interface is designed to be more subtle: the user can always ignore the model suggestions and treat the interface as a standard layout tool. In fact, in our study, 34% of the suggestive interface layouts used no suggestions, indicating that users often wanted the freedom to create on their own. Note that for the comparison above, we only used layouts where the user had accepted at least one suggestion. Figure 6.8 shows some examples of high and low quality layouts created using each interface.

As we show in Figure 6.7 (right), suggestive layouts received a mean vote fraction of 56.90%±3.20, and the adaptive interface received 63.44%±2.69. t-tests with p < 0.01 indicate that designs created by the new interfaces are overall preferred to those from the baseline. It is worth noting that a skilled novice designer can still make a better layout with the baseline interface than a novice with no design sense using our new interfaces. Our interfaces are not meant to replace the user, but rather to help them explore the space of designs, and to make small improvements. Furthermore, in our experience running aesthetic studies on Mechanical Turk, there is a great deal of individual variation in aesthetic preferences.

6.5.2 Preference Study

Our second study was an evaluation of user preferences for the different interfaces. We again used Mechanical Turk, and asked workers to use both the baseline interface (called the “Direct” interface in the study) and a “Suggestion” interface for a single design. Users performed a short tutorial design to familiarize them with the interface, then created two layouts with that inter-
Figure 6.7: Interface evaluation. Using A/B testing on MTurk, we compare layouts created by our suggestive and adaptive interfaces to a baseline layout tool with no suggestions. Left: the count of comparisons where each interface was preferred by 2 or more votes (out of 10 total), or there was a tie. Overall, layouts created with our new interfaces are preferred to those created using the baseline interface. Right: the mean fraction of votes received by each interface in the A/B tests. While the new interface layouts are generally preferred, there is still substantial variation in the layout preferences.

The user then switched to the other interface, completed that tutorial, and created two other layouts. The order of interfaces was randomly flipped. After using both interfaces, workers rated the interfaces and stated their preference (either baseline or the new interface). Users could also provide comments. Workers were paid $1.00 and 40 workers completed each paired test.

There was no statistical difference between the ratings for the different interfaces. The mean rating for the baseline interface was $3.81 \pm 0.19$, with a median rating of 4. The mean rating for the new interfaces was $3.86 \pm 0.26$ and $3.67 \pm 0.30$ for the adaptive and suggestive, with a median rating of 4 for both. Users were split fairly evenly in their preference for the baseline versus the new interfaces. 60% of users preferred the baseline to the adaptive interface, and 59% of users preferred the baseline over the suggestive. To investigate further, we next examine some of the qualitative feedback.
6.5. Evaluation

Figure 6.8: Example user layouts. We show examples layouts created with each interface. The top two rows are higher-rated layouts, and the bottom two rows are lower-rated layouts. The score for each is the mean fraction of votes the layout received during A/B testing.
### Qualitative Feedback

In both studies, users were encouraged to provide qualitative feedback on the interfaces. Note that for the suggestive and adaptive modes, the interfaces were both called “Suggestion,” whereas the baseline interface was called “Direct.”

Many users reported enjoying the new interfaces: “I really like this interface and how it automatically lines things up.” “I really like the brainstorming section and how when you move elements, the font adjusts itself. Very cool and fun to use,” particularly the brainstorming suggestions: “I liked the suggestion interface, it gave you a good starting point to work from whereas the direct you had to start from scratch. The suggestion gave you something to dive off from,” “I preferred the suggestion interface because it gave me new ideas on how to do the layout,” “Great to have a decent starting point,” “The suggestion interface acted as a supplement to my own thought. I could find creativity and inspiration in it.”

However, a number of users disliked the automatic shifting of elements in the adaptive interface: “I did not like when the picture would resize the words or pictures without my permission.” “Interface auto-correction too much, should not be as quick to align everything.” “The other one jumped around and I didn’t like that.”

Many users who preferred the baseline interface did so because of the higher-level of control, or that the suggestions were poor: “I liked having a lot more control over the elements with the Direct Interface.” “I liked having more control. I’m...pretty comfortable handling everything myself. It just felt more natural.” “I preferred the direct to the suggestive, because the suggestions were not very good.” Furthermore, a number of users disliked even having suggestions: “I feel I am more creative if I use my ideas,” “I felt as if the suggestions were taking the human element out a little,” “Typically I already have an idea for how I want to do things. So I don’t really pay too much attention to the suggestions. I’d honestly rather come up with ideas on my own.”

These results are surprising, though understandable given the context of Mechanical Turk. Many Human Intelligence Tasks (HITs) on Mechanical Turk are quite tedious, and require little creativity. A number of users reported enjoying our HITs, and wished they could do more. For example, “I really enjoyed making these, very creative” and “Very fun HIT!”

Another interesting problem is that the model-based suggestions can constrain the possible design. One user reported, “I preferred the direct interface because I felt like the suggestions were so good, I did not feel I could make something significantly different. Losing the suggestions allowed me to ‘think outside the box.’...I’m pretty sure using the suggestion interface first caused me to have some conceptions about my designs.”
6.6 Discussion

In this chapter, we have presented a novel system for graphic design using layout suggestions. Our key contribution is not the energy-based model, but rather the interface and interaction techniques that allow suggestions to aid the design process. In particular, our adaptive interface allows users to create layouts using automatic improvements in a fluid manner. We also evaluate a suggestive interface where users must actively accept changes. We find that users create better designs using these interfaces than a baseline interface without suggestions.

However, we also find that automatic suggestions are not always desired by users. Many users preferred to have complete control over the design process, and found that automatic suggestions took away from their creativity. This result suggests further research is required to help make suggestion-based interfaces more acceptable to users, particularly for creative tasks like graphic design.

The studies presented here are also fairly limited. All the designs were created and evaluated by novices on MTurk. Evaluation of the interface and results from professional designers may help validate our approach. In-person user studies are also required to further investigate how people use automatic design suggestions.

In Fig. 6.9 and the project’s Supplemental Video\(^1\), we demonstrate the system on a tablet. Suggestion-based tools are well-suited for touch interfaces where precise control of elements is difficult.

Current tools often make the design process tedious and time-consuming, particularly for

---

\(^1\)Available at www.dgp.toronto.edu/~donovan/layout
novices. In this chapter, we have presented a small step towards helping users create design layouts. However, many open problems remain. Our brainstorming suggestions are naively displayed in a scrolling list. Improved interfaces for exploring these alternatives, perhaps using attributes, is one avenue of future work. Improving the model to have more stylistic variety is also important. Collaboration tools that allow users to share and refine designs with other people is another possibility.
Exploratory Font Selection Using Crowdsourced Attributes

Geometry can produce legible letters but art alone makes them beautiful. Art begins where geometry ends, and imparts to letters a character transcending mere measurement.

– Paul Standard

Typography is what language looks like.

– Ellen Lupton

Typography is fundamental to graphic design. A well-chosen font can make a design more beautiful and more effective in communicating information. Font selection is also subtle: many professional designers take entire courses on typography, and, for novices, the process can be frustrating and opaque. Surprisingly, the standard interface for selecting fonts — a long list of font names — has changed fairly little over the decades, with only incremental improvements such as font previews, or manually defined font categorizations. These long lists of fonts often overwhelm users with too many choices and too little guidance, and as a result, users often proceed with the default font, or stick with a few familiar, but poor, choices.

The problem of font selection is challenging for many reasons. First, the space of possible fonts is quite large. Most computers are now equipped with hundreds of fonts, and online repositories provide thousands more. Second, there is no obvious method for categorization

---

A version of this chapter was published in ACM Transactions on Graphics (Proc SIGGRAPH), 2014. [113]. This paper was a collaboration between the author, Jānis Libeks, Aseem Agarwala, and Aaron Hertzmann. The project page is at www.dgp.toronto.edu/~donovan/font. The Supplemental Video for this paper can be viewed there, as well as links to the interfaces.
that supports a user's goals. Modern font listings use categorizations like “serif” or “display,” but these must be hand-annotated, and they don't necessarily correspond to a user's goals. Font names themselves are rarely meaningful. Third, there are a variety of font selection tasks with different goals and requirements. One designer may wish to match a font to the style of a particular image. Another may wish to find a free font which looks similar to a commercial font such as Helvetica. A third may simply be exploring a large set of fonts such as Adobe TypeKit or Google Web Fonts. Current methods for font selection fail to address any of these needs well. Exhaustively exploring the entire space of fonts using an alphabetical listing is unrealistic for most users.

This chapter investigates interfaces for selecting fonts based on the idea that fonts can be modelled by attributes: adjectives that describe their visual personality or appearance, such as “formal,” “friendly,” or “legible.” We use these attributes as the basis for three font selection interfaces that are designed to support different types of exploration and search. First, we describe an Attribute Interface that allows a user to select one or more descriptive attributes; the system then shows a list of fonts that are sorted by how highly they score along the selected axes, e.g., we can show fonts that are both friendly and legible. Second, we propose a Group Interface that shows the user a hierarchical menu of fonts, clustered according to their visual similarity. Third, both interfaces include a Search-By-Similarity feature which gives a list of fonts sorted by similarity to their currently-selected font, allowing users to fine-tune their choices. These interfaces allow users to search in a variety of ways, from high-level exploration using attributes or font groups, to refinement of font choices using the similarity search.

In this work, we propose an approach that estimates attribute values and visual font similarity by learning models from crowdsourced data. We first collect a set of attributes commonly used by designers to describe fonts, and then compute attribute values for a set of fonts using crowdsourced data. We also learn a function that predicts the visual similarity of fonts from crowdsourced data. Finally, we learn to predict attribute values and font similarity from geometric features of fonts, so that our system can handle new fonts without any further data collection.

We evaluate our approach with two real-world design tasks, tested with time-limited in-person experiments and online with Mechanical Turk. First, we test users' ability to find a specific font in an interface, given an image of text in that font. For this task, users are three times more likely to find the font by using either of our interfaces, compared to a conventional list interface. Second, we perform a more subjective test of selecting a good font for a given design. In both experiments, participants are limited to 2 minutes for each task. Our proposed interfaces
show a statistically-significant improvement over the conventional list, though the effect size is small since font choices are highly subjective and multimodal. Participant surveys show that users frequently prefer our interfaces to the conventional interface.

Similarly to the previous chapter, this work emphasizes the interaction between an objectivist model of font attributes, and the individual’s exploration of possible fonts. The system allows a user to explore desired attributes (including objective properties like ‘serif’ or ‘wide’) for a particular design. However, some attributes are certainly subjective; one user’s ‘fun’ font may be another’s ‘bad’ font for example. Exploring subjective perception of font attributes is one area of future work.

7.1 Related Work

To our knowledge, no prior work directly studied font selection interfaces, though there is work in a number of related areas.

Font Synthesis and Exploration Metafont system [62] users to create fonts programmatically, using a language for defining vector-based fonts. Metafont’s parametric approach also enables interactive editing interfaces, for example, the Metaflo system [103]. Shamir and Rappoport [140] present an interactive system for font synthesis where a user can manipulate typographic features like stroke width, height, and angles, to easily create font variations. Campbell and Kautz [14] learn a continuous manifold of fonts from examples, allowing a user to interactively navigate possible fonts, including those extrapolated from the training data. In contrast to these systems, we focus on font selection for a particular design, not general font creation.

Font Traits and Features. Researchers have studied the personality traits of certain typefaces and how they affect the appropriateness of typefaces for specific tasks. Shaikh [138] provides an extensive list of typeface personality attributes that have been used in academic publications since 1920. Li and Suen [79] and Mackiewicz and Moeller [87] present small scale studies that explore the range of personality values for tens of fonts using a Likert scale. Shaikh et al. [139] performed an online study with hundreds of participants where 20 fonts were rated for 15 adjective pairs (i.e., “sad/happy”) and design type appropriateness. Users consistently attributed personality traits to fonts, and felt specific document types were more appropriate for certain fonts. For example, children’s documents were seen as more appropriate for “funny” fonts. Lewis and Walker [77] showed that when a word’s meaning does not match its font personality traits, participants take longer to analyze the word. These results suggest that attributes are useful in font interfaces. We differ from prior work by developing a working attribute-based interface. We
also use data from a much larger study (31 attributes and 200 fonts) to train predictive models.

Optical Character Recognition systems use low-level raster features for text recognition [145, 144]. We use vector features, which are not available in the OCR setting, and should be higher-quality indicators of style.

Another approach for representing fonts is HP’s PANOSE standard [75], which assigns a set of category numbers based on visual characteristics like weight or serif style. Proprietary mapping software can then be used to search by similarity. Unfortunately, there are no automatic methods of classifying fonts with PANOSE numbers, with Doyle [25] reporting an adoption rate of less than 10% in his analysis of the system. We instead use an empirical approach to measure font similarity, and train models based on crowdsourced data.

**Commercial Font Interfaces.** While the majority of font selection interfaces in existing applications are simple lists, more sophisticated approaches have been developed. Several websites allow searching by font similarity, including Identifont, MyFonts’ WhatTheFont, and Fontspring. The TypeDNA Photoshop plugin allows searching by similarity, as well as 4 attributes (“weight,” “width,” “italic,” and “optical”). Unfortunately, the details of these searching algorithms are proprietary and may be partly based on hand annotation, so it is difficult to directly compare with our approach. Sites like Fonts.com and Dafont have a large number of attributes and categories for fonts, including subjective attributes. Unlike our automatic approach however, these binary labels are hand-annotated.

**Attributes.** Object attributes have recently become an active topic in computer vision [70, 152, 122], and other domains such as music [88], and even color themes [19]. While binary attributes have been used for image search [70, 152], recent work for estimating relative attributes [122] has been used in search interfaces. Our work is similarly inspired by this approach.

Chaudhuri et al. [15] use relative attributes to help users find parts to assemble into 3D models. Though their focus is 3D modelling, they also show a proof-of-concept web design interface where page elements such as fonts or background color are automatically swapped using an attribute slider. The system also uses font features, including size and Fourier coefficients to measure shape. However, their dataset only included 30 WordPress templates, and their attribute model and interface were not evaluated in any way. By contrast, our work focuses on font selection, uses a much larger dataset, and includes a rigorous evaluation.

WhittleSearch [66] is especially similar to our work as it allows searching image collections using relative attributes. However, we simplify the interaction considerably, as well as introduce cluster and similarity-based interfaces that support fluid switching between different types of search.
Exploratory Search. Interfaces that support exploratory search through large datasets are a common topic for the HCI and information retrieval communities [165]. One approach is to visualize high-dimensional data points plotted within a low-dimensional embedding such as t-SNE [159]. We experimented with a t-SNE interface, but found it difficult to interpret the 2D layout; it is hard to know where to look in this space for a particular font, particularly with thousands of fonts. Another approach that we take is to create hierarchical categorizations of the data; for example, Huang et al. [50] hierarchically cluster 3D shapes based on a hand-designed distance metric. A related problem is the exploration of continuous parametric spaces for design [92, 150]; our approach is more appropriate for exploring a discrete design space. We also experimented with an adaptive grid interface, similar to Marks et al. [92], where users could select several fonts and the system would adaptively display similar fonts. In practice this approach was far more time-consuming than our group interface; numerous selections were required to converge on reasonable fonts, and the approach did not provide a high-level view of the entire font space.

7.2 User Experience

We first describe our interfaces from the perspective of a user (a demonstration video is available at the project website). We offer two ways to begin exploring fonts, as well as a Search-by-Similarity option to fine-tune an initial font selection.

Attribute Interface. The attribute interface is useful when a user has a conceptual rather than mental image of the desired font. For example, a user may want “happy” and “playful” fonts for a child’s birthday card, “formal” fonts for legalese, or “legible” fonts for a wall sign. The interface uses a menu listing the set of attributes available (Figure 7.1a). The user selects an attribute by clicking it, or by clicking on the adjacent “not” button to select fonts that do not exhibit an attribute (e.g., picking “not strong” produces “weak” fonts). When the user mouses over an attribute, five example fonts with that attribute are shown as examples, as well as five fonts lacking that attribute. Once the user selects an attribute, or a combination of attributes, fonts are shown sorted in order according to how well they match the selected constraints (Figure 7.1b). Multiple attributes may also be selected, in which case fonts are sorted by the average of how well they match the selected attributes.

Group Interface. The group interface supports a more visual exploration of the space of fonts

---

1The project page is at www.dgp.toronto.edu/~donovan/font
Figure 7.1: **Attribute interfaces.** The interfaces are shown next to the graphic being designed so that users can see font choices in context. (a) Attribute selection menu. The user may select one or more attributes and/or complements. When mousing over an attribute, examples of fonts with and without the attributes are shown on the right. (b) As the user selects attributes, a list of fonts with the given attributes are shown in the list; here, the user has selected delicate and not thin.

for users who will recognize desirable fonts on sight. The group interface organizes the space of fonts into a tree-based hierarchy of visually similar fonts (Figure 7.2). That is, the leftmost list contains twelve groups of visually similar fonts, with a representative sample shown from each. When the user mouses over a font, the middle list shows subgroups of that font’s group. Each subgroup is shown by a representative font. Mousing-over any of these reveals their subgroups. The user clicks on a font in any list to select it.

The groups are created automatically with a bottom-up hierarchical clustering algorithm. All font clusters are distinct, that is, fonts are restricted to a single cluster. The unique assignment is performed to minimize the overall number of fonts in the menu. However, once a user selects a font, all similar fonts are displayed, regardless of the cluster, so the unique assignment does not hinder the user’s ability to explore the overall set of fonts.

**Search-by-Similarity.** Both of the above methods are useful for quickly identifying a reasonably appropriate font. Once a user has made an initial choice with either of the above methods, the Search-by-Similarity option (Figure 7.3) helps the user refine their choice. This option lists
Figure 7.2: Group interface. The interface shows a three-level perceptual clustering of fonts. Mousing over the clusters allows a user to quickly get a sense of the range of options, and to explore individual clusters. Once a user has selected a font, they may further refine their query by searching for similar fonts as in Figure 7.1.

fonts ordered by their visual similarity to the selected query font. Users can replace the query font with another by pressing a button next to any selected font. They may again show the most similar fonts to the new selection, thus exploring the space of fonts “near” their current favorites.

For all three interfaces the current design is shown to the right, and the user may choose between three text sizes (“small,” “medium,” and “large”); users can also save “favorite” fonts to a separate list. We display all fonts with a word “handgloves” that is often used by typographers for font comparison [32] since it contains many different letter strokes and shapes. Removing font names from the interface also reduces familiarity biases during evaluation.

To support these interface tools, we require models for relative attributes and font similarity. We next describe our approach for training these models from crowdsourced data.

7.3 Estimating and Predicting Font Attributes

This section describes our technique for estimating and predicting relative font attributes. We gather pairwise comparison data by asking Mechanical Turk workers to compare a small set of training fonts according to different attributes. We then estimate relative scalar values for each training font and attribute, and use these values to train a model that maps from fonts to
attributes. We then compute attribute values for a much larger font database using this model. We gather data from novices on MTurk instead of professionals because novices are the target users of our interfaces.

### 7.3.1 Font Selection

We gathered a large and diverse set of 1278 fonts that combines 1138 fonts from Google Web Fonts with a selection of web fonts that appear frequently on a relatively small set of ≈3800 design-oriented web pages, seeded from the Adobe Typekit Blog. Typefaces within the same font family often have very different personalities, and so we treat each separately, e.g., Gill Sans is treated as a separate font from Gill Sans Light and Gill Sans Bold. We then randomly sampled a training set of 200 fonts for the MTurk experiments.

### 7.3.2 Attribute Selection

We chose 31 attributes from a list of font personality attributes gathered by Shaikh [138], reflecting adjectives that we expect novice users would use to describe fonts. We included con-
crete attributes such as “thin” and “angular,” and more nebulous concepts like “friendly” and “sloppy.” We also added 6 common typographical binary attributes: capitals, cursive, display, italic, monospace, serif. All 37 attributes are listed in Fig. 7.1. Relative attribute values range from 0 to 100, whereas binary attribute values may be either 0 or 100. We hand-label the 6 binary attributes in the training set, leaving the relative attributes to be estimated through crowdsourcing. We also performed an earlier version of this study with 36 relative attributes, and then pruned five after finding high correlations with other attributes. For example, “masculine” and “strong” were highly correlated, so “masculine” was removed.

### 7.3.3 Attribute Estimation

The goal of estimation is to determine a scalar value describing how much a given font embodies a given attribute. For example, a font that is often considered “stronger” than other fonts should have a higher value for the “strong” attribute. Directly asking people to provide these scores would be unreliable. Instead, we follow a standard approach and ask comparison questions. A Mechanical Turk worker is shown a pair of fonts, and asked to rate which is better described by the attribute (Figure 7.4). We use a two-alternative forced choice (2AFC) design, i.e., raters cannot answer “no difference,” in order to allow us to better measure small differences.

Given these pairwise comparisons, we can then estimate the attribute value for each font. We use a Maximum Likelihood approach to attribute value estimation using the Bradley-Terry model [10]. See the survey by Tsukida and Gupta [158] for details. However, we augment it with a model of rater reliability, similar to item-response theory and the work of Welinder et al. [163].

The measured pairwise responses form a set of tuples $\mathcal{D} = \{(a, f_i, f_j, u, q)\}$, where $a$ is an
attribute, $f_i$ and $f_j$ are the two fonts being compared, $u$ is the rater’s ID, and $q$ is the rater’s choice. $q = 1$ if the rater judges font $f_i$ to have more of the attribute $a$ than font $f_j$, $q = 0$ otherwise. In the standard approach, the likelihood of a rater’s response $q$ given the fonts and attribute is modeled as follows. Let $v_{i,a}$ and $v_{j,a}$ be the unknown values of attribute $a$ for the two fonts. A rater is more likely to answer $q = 1$ if $v_{i,a} > v_{j,a}$, and $q = 0$ otherwise. However, the rater’s response is more random (harder to predict) if the difference in attribute values is small. In the extreme case where $v_i = v_j$, the rater’s response is entirely random ($p(q = 1) = 0.5$). To model the rater’s response, we use a logistic function:

$$p(q = 1| f_i, f_j, a) = \frac{1}{1 + \exp(v_{j,a} - v_{i,a})}$$  

(7.1)

When performing MTurk evaluations, some raters may be more reliable than others. Hence, we introduce a per-user reliability weight $r_u$. Raters with low $r_u$ produce more random answers; raters with $r_u = 0$ are completely random, and raters with $r_u < 0$ tend to produce wrong answers. In the reliability model, the likelihood of a comparison is:

$$p(q = 1| f_i, f_j, a, u) = \frac{1}{1 + \exp(r_u(v_{j,a} - v_{i,a}))}$$

(7.2)

Figure 7.5: Examples of estimated attribute values. We show the fonts with the least of the attribute ($v = 0$), most ($v = 100$), and intermediate values ($v = 33, 66$).
Given the pairwise comparisons $D$, the negative log-likelihood objective function is:

$$E(v, r) = -\ln p(D|v, r)$$

$$= - \sum_k q^k \ln p(q = 1| f_i^k, f_j^k, a^k, u^k)$$

$$- \sum_k (1 - q^k) \ln \left(1 - p(q = 1| f_i^k, f_j^k, a^k, u^k)\right)$$

where $k$ indexes over all training tuples. We jointly minimize this objective with respect to all attribute values $v$ and all rater reliabilities $r$ by gradient descent. The resulting attribute values are scaled to lie in the range 0 to 100. We gathered the comparison data by a large-scale study on Mechanical Turk, described in Appendix F. Figure 7.5 shows examples of the estimated values.

We find that raters agree more on certain attributes than others. In Table 7.1 we list the error in modeling the pairwise comparisons for each attribute. These results show which attributes are easiest to model (“thin,” and “strong” for example), as well as the most difficult (“sharp” and “boring”). In Fig. 7.6 we show histograms of 4 different attributes. These histograms show that the attributes being learned have different distributions.

In Fig. 7.7 we show a histogram of different user weights. Most users had positive weights, though a few users had negative weights, indicating they generally provided answers opposite to the majority opinion. To compute font attributes, we used these users with negative reliability weights, on the assumption that the users may have misunderstood the directions. While we use these users for our final attribute estimation, given their small number, we could also discard these invalid users. We also plot the user reliability weights against the number of tasks completed by the user. This figure shows that the number of tasks completed is not correlated with the user’s reliability.

### 7.3.4 Attribute Prediction

We now describe an approach to learning a mapping from font features to font attributes, using the estimated attributes as training data. The features, denoted $x_i$ for font $i$, are computed from font files using the raw glyph outline control points and points sampled from the glyph outline curves. Features were selected in part to capture typographic font qualities (italics, thickness), as well as other vector-based qualities. We include features which measure the size, area, orientation, stroke width, and spacing of characters. We include vector-based features such as curvature, number of curves per glyph, arc lengths, etc. See Appendix E for a detailed description.

We use these features to learn separate models for each of our 37 attributes. We learn the attribute values using Gradient Boosted Regression (GBR) Trees [28] with a maximum depth of 10.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Mean Log-Likelihood</th>
<th>Classification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>thin</td>
<td>0.181</td>
<td>93.16%</td>
</tr>
<tr>
<td>strong</td>
<td>0.352</td>
<td>84.80%</td>
</tr>
<tr>
<td>wide</td>
<td>0.422</td>
<td>81.11%</td>
</tr>
<tr>
<td>soft</td>
<td>0.501</td>
<td>75.86%</td>
</tr>
<tr>
<td>disorderly</td>
<td>0.507</td>
<td>73.88%</td>
</tr>
<tr>
<td>artistic</td>
<td>0.509</td>
<td>73.80%</td>
</tr>
<tr>
<td>complex</td>
<td>0.520</td>
<td>73.09%</td>
</tr>
<tr>
<td>playful</td>
<td>0.537</td>
<td>71.75%</td>
</tr>
<tr>
<td>calm</td>
<td>0.548</td>
<td>70.80%</td>
</tr>
<tr>
<td>dramatic</td>
<td>0.549</td>
<td>71.08%</td>
</tr>
<tr>
<td>sloppy</td>
<td>0.559</td>
<td>70.20%</td>
</tr>
<tr>
<td>clumsy</td>
<td>0.563</td>
<td>70.14%</td>
</tr>
<tr>
<td>bad</td>
<td>0.570</td>
<td>68.53%</td>
</tr>
<tr>
<td>attention-grabbing</td>
<td>0.573</td>
<td>69.23%</td>
</tr>
<tr>
<td>formal</td>
<td>0.576</td>
<td>69.44%</td>
</tr>
<tr>
<td>attractive</td>
<td>0.579</td>
<td>68.08%</td>
</tr>
<tr>
<td>legible</td>
<td>0.580</td>
<td>68.28%</td>
</tr>
<tr>
<td>charming</td>
<td>0.586</td>
<td>67.67%</td>
</tr>
<tr>
<td>gentle</td>
<td>0.601</td>
<td>66.19%</td>
</tr>
<tr>
<td>modern</td>
<td>0.601</td>
<td>65.86%</td>
</tr>
<tr>
<td>pretentious</td>
<td>0.611</td>
<td>66.02%</td>
</tr>
<tr>
<td>happy</td>
<td>0.611</td>
<td>65.03%</td>
</tr>
<tr>
<td>graceful</td>
<td>0.619</td>
<td>65.05%</td>
</tr>
<tr>
<td>fresh</td>
<td>0.619</td>
<td>64.16%</td>
</tr>
<tr>
<td>friendly</td>
<td>0.622</td>
<td>64.39%</td>
</tr>
<tr>
<td>delicate</td>
<td>0.623</td>
<td>63.83%</td>
</tr>
<tr>
<td>warm</td>
<td>0.630</td>
<td>64.53%</td>
</tr>
<tr>
<td>technical</td>
<td>0.634</td>
<td>62.94%</td>
</tr>
<tr>
<td>angular</td>
<td>0.635</td>
<td>62.80%</td>
</tr>
<tr>
<td>boring</td>
<td>0.636</td>
<td>63.05%</td>
</tr>
<tr>
<td>sharp</td>
<td>0.649</td>
<td>62.02%</td>
</tr>
<tr>
<td>average</td>
<td>0.558</td>
<td>69.52%</td>
</tr>
</tbody>
</table>

Table 7.1: Estimation of relative attributes. We report the negative log-likelihood (lower is better), and the classification rate, the fraction of pairwise comparisons correctly predicted (higher is better)

2, and also a linear LASSO model [155]. On leave-one-out cross-validation tests, the GBR had better performance with a mean average error of 8.51 compared to LASSO’s 10.76; we therefore use this model for all further tests.
7.3. Estimating and Predicting Font Attributes

7.3.5 Model Evaluation

An alternative approach is to use the pairwise comparison data to directly train the mapping from geometric features to attributes. The method of Parikh and Grauman [122] takes this approach by using a ranking SVM (SVMrank). We also modify our method to take this approach by learning weights for a distance of geometric features rather than attributes. This Feature Weight model is a simple extension of the method we describe in Section 7.3.3 to estimate attributes. In that section, we model the probability $q$, that a rater judges font $f_i$ to have more of the attribute $a$ than font $f_j$. We represent this probability using a sigmoid:

$$p(q = 1| f_i, f_j, a, u) = \frac{1}{1 + \exp(r_u(v_{j,a} - v_{i,a}))}$$  \hfill (7.5)

Where $v_{i,a}$ and $v_{j,a}$ be the unknown values of attribute $a$ for the fonts $i$ and $j$, and $r_u$ is a per-user reliability values. These values are then estimated using gradient descent.

In the Feature Weight model, the values are computed by taking the product of fixed font features $x$ with a learned per-attribute weight vector $w_a$:

$$p(q = 1| f_i, f_j, a, u) = \frac{1}{1 + \exp(r_u(x_jw_a - x_iw_a))}$$  \hfill (7.6)
Figure 7.7: **Left**: histogram of user reliability weights assigned by fitting the objective function. **Right**: plot of user reliability weights with respect to the number of Human Intelligence Tasks (HITs) (log scale) with the median weight indicated by a green horizontal line. Note that some user weights are negative, implying that the user is judged to have chosen the opposite answer of what attribute values would predict. There is no correlation between the number of completed tasks and the user reliability.

To evaluate the methods, we repeatedly train a model on the comparisons for 199 fonts and test on the hold-out font comparisons. We report the negative log-likelihood (NLL) of the test data and the classification rate (the fraction of comparisons correctly predicted). We also report an upper-bound “oracle” classifier which chooses the majority opinion. Table 7.2 shows that all methods perform similarly and do quite well, considering the high level of user disagreement.

When comparing the models, the likelihood is a more accurate continuous measure of performance as it uses the attribute distances. For example, the classification rate for two attribute values would be the same if the two attributes are very close or far apart, as long as the relative ranking is correct. Intuitively, such a discontinuous measure is not ideal since we expect that when the attribute distances are very small, users will have a harder time ranking them correctly. However, user modelling only results in a slightly lower NLL, and no difference in the final classification rate, suggesting that the user reliability weights are not particularly important for attribute estimation.
### Table 7.2: Comparison of attribute prediction models.

Our Feature Weight method learns a set of linear weights on font features to match the pairwise comparisons. Our Direct Attribute method estimates the attribute values independently of features, and then trains a GBR using features for generalization. We show results without user reliability modelling, denoted as “w/o user.” To evaluate the algorithms we report the negative log-likelihood (NLL) for testing data (lower is better) and the classification rate of the pairwise comparisons. NLL data is unavailable for the Oracle classification algorithm, which classifies a comparison as correct if it matches the majority opinion, as well as the SVMRank algorithm [122], which uses a non-probabilistic objective function. All methods perform equally well at the task. User modelling does result in a slightly lower NLL, but no difference in the final classification rate, suggesting that the user reliability weights are not particularly important for attribute estimation.
7.4 Font Distance Metric

We next describe an approach to learning to predict the perceptual similarity between fonts, which is required for the Search-by-Similarity tool. A naive similarity metric would simply use the Euclidean distance between vectors of geometric font features, or between vectors of the 37 estimated attributes. However, it is unclear which attributes or features are more or less important when people evaluate the overall perceptual similarity. For example, it is possible that a difference in the “thin” attribute is more indicative of a visual difference than “serif.”

We next describe the problem formulation and related work in metric learning, and then describe the results of our similarity study using MTurk. Our results show that font attributes outperform geometric font features for modelling font similarity, demonstrating their usefulness as a mid-level representation for fonts.

7.4.1 Metric Learning

Learning distance metrics is a well-studied problem in the machine learning community [69]. Distance metric learning aims to create distance function between objects; we refer to the distance between object $i$ and $j$ as $d_{i,j}$. Typically, objects are embedded in some feature space $x_i$; metric learning attempts to create a new embedding whose distances better matches a set of input distance constraints. For example, we might specify relative distances in the new embedding space using triplets (i.e, $d_{i,j} > d_{i,k}$).

Linear metric learning methods learn a matrix $W$ such that $d_{i,j} = ||Wx_i - Wx_j||$. Schultz and
Joachims [137] use a nonlinear approach based on SVM learning to model distances. Learning these transformations can be posed as a constrained optimization problem which leads to efficient convex solutions. Unfortunately, most of this prior work is not probabilistic and therefore not designed to handle noisy crowdsourced data.

Recently, Tamuz et al. [151] defined a probabilistic sigmoid model for crowdsourced triplet comparisons: \( p_{ijk} = S(d_{i,j} - d_{i,k}) \). Given the triplets, an embedding for each object is learned in a Euclidean space that requires no knowledge of object features. Our probabilistic model is very similar, but we learn a linear embedding matrix for features, thus allowing our approach to extend to unseen objects. We also model the reliability of individual users.

The data we use to train our models is a set of font triplets, expressed in the form \( D = \{(f_i, f_j, f_k, q, u)\} \), where \( f_j \) and \( f_k \) are the two fonts being compared to font \( f_i \), \( q \) is the user’s choice (\( q = 1 \) if the user judges font \( f_j \) is closer to font \( f_i \) than font \( f_k \), \( q = 0 \) otherwise), and \( u \) is the user ID. To model the probability of \( q \), we use a logistic function similar to the one in Section 7.3.3:

\[
p(q = 1|f_i, f_j, f_k, u, W) = \frac{1}{1 + \exp(r_u(d_{i,j} - d_{i,k}))}
\]

where \( d_{i,j} \) is the Euclidean distance between fonts \( i \) and \( j \) in some embedding space, parameterized by \( W \), and \( r_u \) is a per-user reliability weight.

The key question is how to model these distances \( d_{i,j} \) and \( d_{i,k} \). We first consider the simple **unweighted** Euclidean distance of a font feature vector:

\[
d_{i,j}^E = \|x_i - x_j\| \tag{7.8}
\]

where \( x_i \) is a feature vector for font \( f_i \) with size \( n \). In this model, we use either the geometric features of Sec. 7.3.4 (\( n = 80 \)), or the predicted attribute vector (\( n = 37 \)). We evaluate both feature sets in the next subsection.

The second model is a **weighted** Euclidean distance:

\[
d_{i,j}^w = \|w^T(x_i - x_j)\| \tag{7.9}
\]

Where \( w \) is a learned vector of weights.

The third **subspace** model computes the Euclidean distance after embedding the fonts in a lower \( n \)-dimensional subspace:

\[
d_{i,j}^w = \|W(x_i - x_j)\| \tag{7.10}
\]

Where \( W \) is a learned embedding matrix of dimensionality \( m \times n \) where \( n >> m \), where \( n = 37 \) or \( n = 80 \), depending on the feature set used, and \( m = 7 \), as selected by cross-validation.
Given the pairwise comparisons $D$, the negative log-likelihood objective function is therefore:

$$E(M, r) = - \sum_n q^n \ln p(q = 1|f_i^n, f_j^n, f_k^n, u^n, W)$$
$$- \sum_n (1 - q^n) \ln (1 - p(q = 1|f_i^n, f_j^n, f_k^n, u^n, W))$$

We jointly solve for the embedding parameters $W$ and rater reliabilities $r$ using gradient descent.

**MTurk Study.** To obtain data to train the model, we conduct a crowdsourced study focused on font similarity. Workers are presented with a reference font A and two fonts (B and C) and are asked to decide whether B or C is more similar to A than the other. An example task is shown in Figure 7.8. Triplets were randomly sampled from the 200 font training set. See Appendix F for study details including number of triplets, payment, and control questions.

We evaluate the method using leave-one-out cross-validation, in which one font is omitted from training, and then classification is tested on triplets that include the hold-out font; results are averaged over each choice of hold-out font. In Table 7.3 we compare our method with and without user reliability modelling, as well as the SVM approach of Schultz and Joachims [137]. For the SVM model, we found the RBF kernel produced the best results, with the parameters set using cross-validation. The upper bound on the performance is given by an oracle algorithm which always chooses the majority opinion for each triplet. Our probabilistic model outperforms the SVM approach, likely due to the considerable disagreement between users in our data. As in the case of font attribute learning, the user reliability weights result only in a slightly lower NLL, but no change in the final classification rate, suggesting that the user reliability weights are not particularly helpful for this task.

We find that embedding fonts in a subspace model gives the best results (see Table 7.4), though the weighted model also performs quite well. Given the disagreement between users, the subspace model achieves a classification rate of 75.83% on testing data, out of a possible 80.79%. We also report results of the same learning procedure using the geometric features described in Section 7.3.4 rather than attributes; the performance of the different models is similar, though slightly better, with attributes. This result demonstrates that learning a mid-level representation of attributes for fonts is useful for tasks such as modelling similarity.
Table 7.3: We compare our metric learning with and without user reliability modelling, as well as the SVM approach of Schultz and Joachims [137]. We report the negative log-likelihood (NLL) of the test data, as well as the classification rate: the fraction of responses that are correctly predicted. The oracle is the upper-bound on the classification rate, given user disagreement. NLL data is unavailable for the Oracle and SVM algorithms, as they do not output probabilities.

![Table 7.3](image)

Table 7.4: Results of font distance metric learning using different distance models and features. We evaluate using geometric features extracted from the fonts directly (Sec. 7.3.4) as well as our predicted attribute values. We report the negative log-likelihood (NLL) of test font comparisons. We also report the classification rate: the fraction of test responses that are correctly predicted. The oracle is the upper-bound on the classification rate, given user disagreement. The improved performance of the attributes over geometric features demonstrate that attributes are a useful mid-level representation for fonts.

![Table 7.4](image)

7.4.2 Grouping

For the Group Interface, we compute a hierarchical font categorization automatically with $k$-means clustering [85] on fonts in the embedding space. Fonts are clustered in three levels in a bottom-up manner; all fonts are first clustered into 130 clusters, and then the centroids of each cluster are further clustered into 12 groups. The cluster sizes were chosen based on the constraints of a three-tier menu, and a target menu size of 10-12 items. The font nearest the center of each top-level group is shown in the first column of the interface (Figure 7.2).
7.5 User Interface Evaluation

To compare the three interfaces, we conduct user studies on two separate design tasks. In the font-matching task, the user is presented with the image of a font, and attempts to find the font within a user interface. In the design task, the user attempts to select a font that is best suited to a given design. We compare against a baseline list-selection interface, similar to existing applications (Figure 7.9), in which fonts are shown in a random order.

In each trial, the user first reads the instructions for one of the three interfaces, then performs a brief tutorial task to familiarize them with the interface. The user then completes five font-matching or design tasks. The interfaces are reset between tasks. We impose a two-minute time limit on each task, in order to prevent highly-motivated users from exhaustively searching the lists of fonts.

7.5.1 Font Matching Task

We first test users’ ability to find a given font within each user interface (Figure 7.10). Font recognition sites and apps such as MyFonts’ WhatTheFont help users identify fonts used in existing designs, such as signs and advertisements; these sites are heavily used and demonstrate the usefulness of the task. The user might not be able to find the exact font within the two-minute time limit; in this case, the user should seek the most similar font. To simplify the task, the favorites box allows the user to keep a running list of the most relevant fonts for later review. For each
Please make the bottom font match the top font as closely as possible

The quick brown fox jumps over the lazy dog

Figure 7.10: Example of a font matching task. The users use the interface to make the bottom font match the top as closely as possible.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Distance to Target Font</th>
<th>Effect Size</th>
<th>Exact Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>55.08 ± 2.13</td>
<td>-</td>
<td>5%</td>
</tr>
<tr>
<td>Attribute</td>
<td>52.97 ± 2.57</td>
<td>0.07</td>
<td>15%</td>
</tr>
<tr>
<td>Group</td>
<td>47.36 ± 2.20</td>
<td>0.26</td>
<td>15%</td>
</tr>
</tbody>
</table>

Table 7.5: Results of the font matching study. Using the group interface, workers on average selected a closer font than using the baseline and found the exact font more often. For the attribute interface, workers also found the exact target more often, though there was no statistically significant difference for the mean distance. The distance between fonts is computed as the Euclidean distance in the learned embedding space of Sec. 7.4, with 95% confidence intervals.

interface, 10 tasks were created with 5 target fonts each. The target fonts are selected randomly, but workers receive the same sequence of target fonts, regardless of the interface. Additional study details are in Appendix F.

Results from the MTurk study are shown in Table 7.5. We find that, using either of the new interfaces, a user is three times more likely to correctly find the target font, as compared to the baseline interface. We also compare the mean distance of the selected font to the target font for the new interfaces compared to the baseline, according to our learned metric (Section 7.4). The group interface performs significantly better than the baseline interface (using a two-sided t-test between the distances, \( p < 0.05 \)). The effect size was 0.26, computed as \( \frac{|\mu_n - \mu_b|}{\sigma_b} \), where \( \mu_n \) and \( \mu_b \) are the means of the new and baseline interface, and \( \sigma_b \) is the baseline standard deviation. There was no statistically significant difference between the mean distances of the baseline and attribute interfaces (effect size 0.07). When users found inexact matches, they found better matches using the group interface rather than the attribute interface, perhaps because users had trouble identifying appropriate attributes for certain fonts.
Table 7.6: MTurk interface evaluation. Font selections from the new interfaces are compared against the baseline in forced A/B comparisons. We report the raw percentage of people who preferred the design created by the new interface, as well as the percentage of comparisons where one interface had a clear majority (i.e., had 2 or more votes more than the alternative, out of 8 votes total). The new interfaces perform statistically better than the baseline interface, though the effect size is small due to the high variance in user ability and font evaluation. We also compare the baseline interface fonts against the designer’s original fonts, which provides a rough upper bound on the performance. Note the significant noise in the raw results, though designer’s fonts are generally preferred. 95% confidence intervals are shown. The higher confidence intervals for the designer versus the new interfaces is due to a smaller sample size (180 vs. 3600).

### 7.5.2 Design Task

Our second evaluation uses a more subjective task: given a design, pick a suitable font for the main text, such as title or heading. Each design consists of a background image and modifiable text fields, with the background image usually containing text which the user cannot modify. We use 15 relatively simple designs created by three professional designers that reflect a variety of formal and informal event posters, invitations, announcements, and advertisements, similar to designs that an average user might create on their own (Figure 7.11). As with the matching task, workers have two minutes to complete the font selection task on each design.

We conducted the design task using Mechanical Turk, and with in-person studies with local university students. Mechanical Turk workers allow us to gather large amounts of data, whereas in-person studies allow us to perform more in-depth qualitative comparisons.

**MTurk Study.** For each interface we asked MTurk workers to choose fonts for 15 designs. Each study task contained 5 designs and used a single interface, with 1,350 designs collected in total. We then evaluated the designs created using the new interfaces against the designs created by the baseline interface. 2AFC testing was performed with designs selected from the baseline interface and either the attribute or group interface, and evaluated by 8 workers. See Appendix F for study details.

Results are shown in Table 7.6. Our interfaces give a statistically significant improvement.
over the baseline (two-sided t-test with $p < 0.05$). However, the difference in preference effect size is relatively small, perhaps reflecting the highly variable preferences and abilities of MTurk workers at font selection and evaluation. We found considerable disagreement between MTurk raters: average votes had about a 74% majority, where 100% would indicate unanimous vote and 50% would indicate an exact tie.

We also compare the MTurk font selections to those originally chosen by the professionals that created each design (Table 7.6), using the same 2AFC study design as above. Each designer’s font selection was compared to 12 baseline interface font selections, with random comparisons added to prevent a familiarity bias. This comparison provides a rough upper bound on the performance of new interfaces, as the designers selected the fonts when creating the original design, so the font should be a good choice for the design. Remarkably, the designer font choices were preferred over MTurk selections only 53.19% of the time, suggesting there is substantial noise and subjective preference in the evaluation.

We can also estimate a relative score using pairwise comparisons in the same manner as in Section 7.3.3. Figure 7.11 shows an example of the best, worst, and middle font selections for 4 of the 15 designs. We also include the scores of the original designer font choices.

**Designer vs. MTurk Evaluation.** The high level of disagreement when evaluating fonts makes it unclear whether using novices on MTurk is appropriate. We therefore conducted a study comparing font evaluation between MTurk users and three professional designers, recruited online.

In the previous section we use pairwise A/B comparisons between fonts to evaluate, and estimate rankings from multiple MTurk workers. However, this approach is not appropriate for directly comparing novices and designers. To allow a simpler comparison, we created ranking tasks where users were shown 9 font selections and ranked them from best to worst. The professional designers were paid $20 an hour, and completed 104 ranking tasks. MTurk users were paid $0.07 for each ranking task, and could complete as many ranking tasks as they desired.

Due to the subjective nature of font evaluation, there is no correct ranking. However, given the experience and training of professional designers, it is expected that their responses are more trustworthy than novices. To compare these two groups, we use Kendall tau rank correlation coefficient. A score of 0 indicates no correlation between rankings, and a score of 1 indicates an exact agreement between rankings. The final Kendall tau score is found by averaging over all comparisons over all ranking questions. To control for the higher number of MTurk workers, for each ranking task, we selected the response of three workers at random.
Figure 7.11: Examples of the estimated design scores from crowdsourced pairwise comparisons. The top three rows show MTurk designs ranging from 0 (worst) to 100 (best). The bottom row show the original designer font choices, along with the estimated relative score.
We first compute the intra-group Kendall tau scores to measure how consistent the group members are. We found the consistency of users within both groups were similarly low, with mean of the absolute values of the Kendall tau coefficients of 0.369 and 0.365 for MTurk and designers respectively. The Kendall tau coefficient between the two groups was 0.322. These results suggest that font evaluation is highly subjective, for both novices and professionals. Furthermore, the agreement between novices and professionals is only slightly lower than between professionals themselves, suggesting MTurk evaluations are reasonable.

**Qualitative MTurk Evaluation.** Our tasks also included fields for providing general comments and suggestions on the task and interface. We received many positive comments, including “I thought the interface was easy to use. It was helpful in picking out fonts based on attributes that you thought the design should have (i.e. “happy” for birthday card)”, “This would be an EXCELLENT tool for setting typefaces. Amazingly easy to use/edit. Fun!”, “The attribute selector was very helpful!”, “Its a great concept...People like me who are not pros would love this. Even professionals would love this.”, “This actually seems like a really neat tool. As someone who occasionally needs choose a font I’m often overwhelmed with the flat list of choices. I look forward to finding this out in the wild.”

Some users also provided suggestions or negative comments. Most comments were related to task constraints (e.g., requesting more time for the task), or requesting more features (e.g., selecting bold or italic fonts, increasing the sizes of certain interface elements). A few users wished for the font groups to be labelled.

**In-Person Testing.** To further evaluate our interfaces, we also conducted an in-person study with 31 participants; 17 were second-year design students and the rest were recruited from a study participant mailing list. Each participant used all three interfaces in shuffled order, creating 5 designs with each interface, with a time constraint of 2 minutes per design. After the font selection, each participant rated each interface based on various factors (overall preference, ease of use, ease of learning), and commented on each interface. We also showed participants all 15 designs in random order and asked them to rate their satisfaction with the font selection.

Table 7.7 shows the mean ratings for each interface. Participants generally preferred the new interfaces to the baseline interface, though the difference between the group and baseline interface was not statistically significant (using the Mann-Whitney U Test). As expected, participants found the list interface easiest to learn, given its similarity to existing font selection interfaces.

Comments on the attribute interface were mostly positive, with 23 participants enjoying the interface, including: “This was a great interface. I like that I had the option to select or not select attributes based on how I was feeling about the design.”, “I liked this interface the best. When
choosing a font, I thought of a few words that would describe my objective for the poster. After selecting my attributes, the ‘similar fonts’ button allowed me to find a few more fonts that suited my needs.” However, the interface took longer to learn: “This is tricky without knowing what the attributes are initially, but quite handy after multiple uses” and “was a little overwhelming at first.” One participant felt the attributes “were too broad. ‘Artistic’ can capture a number of ideas.”

Comments on the group interface were more mixed, with 15 participants enjoying the interface: “This was the easiest to use, as it knew exactly which one I am going for.” However, other participants found the interface complex: “If I chose one font, there were a lot of other fonts to choose from so it was confusing,” and found the groups hard to interpret. This interface does present the steepest learning curve of the three; it is possible that once users were familiar with the menu selection, these issues would be diminished. The font grouping could also be refined to remove redundant groups, or further organized by an expert.

Comments on the baseline interface were mostly negative, with 20 of the participants mentioning the difficulty of dealing with a large number of fonts: “Definitely the most time consuming and irritating of the bunch since I had to scroll through a lengthy list just to find a specific font.” However, some participants did prefer the simplicity: “This interface is okay if the list of possible fonts were not too large....the interface was very simple to learn.” It is worth noting that if the design task is an extremely simple one, such as choosing a font for an essay, then an exploratory interface is not appropriate. Users would be better served by a small list of high-quality fonts.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Design Satisfaction</th>
<th>Ease of Learning</th>
<th>Ease of Use</th>
<th>Overall Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>$m: 4, \mu: 3.79 \pm 0.15$</td>
<td>$m: 4, \mu: 3.87 \pm 0.40$</td>
<td>$m: 3, \mu: 3.35 \pm 0.41$</td>
<td>$m: 3, \mu: 2.81 \pm 0.33$</td>
</tr>
<tr>
<td>Group</td>
<td>$m: 4, \mu: 3.70 \pm 0.16$</td>
<td>$m: 4, \mu: 3.42 \pm 0.31$</td>
<td>$m: 3, \mu: 3.58 \pm 0.38$</td>
<td>$m: 3, \mu: 3.07 \pm 0.33$</td>
</tr>
<tr>
<td>Attribute</td>
<td>$m: 4, \mu: 3.76 \pm 0.18$</td>
<td>$m: 4, \mu: 3.60 \pm 0.34$</td>
<td>$m: 4, \mu: 4.03 \pm 0.32$</td>
<td>$m: 4, \mu: 3.81 \pm 0.36$</td>
</tr>
</tbody>
</table>

Table 7.7: Interface ratings: median ($m$) and mean ($\mu$) with 95% confidence intervals. After using all three interfaces, users were asked to rate the ease of learning, use, and overall preference for each interface. Users were shown their design choices from all three interfaces, in random order, and asked to rate their satisfaction with the font selection. There was no significant difference in final design satisfaction between the interfaces, though users preferred the attribute interface and found it easier to use than other interfaces.
7.6 Discussion

In this chapter, we have proposed interfaces for font selection based on estimating attributes of fonts through a combination of crowdsourcing and machine learning. While we focus on fonts, this approach can extend to any domain where users must search large datasets. Potential domains include vector illustration (e.g., a sketchy drawing of a car), music (e.g., playful electronic music), and videos (e.g., a cute video of a dog).

Font attributes and font similarity are inherently subjective properties; however, there is enough agreement among human opinions to build reasonably effective models of these properties. Nonetheless, one limitation of our approach is that it learns the averaged response over all users, and ignores variation among individuals. Modelling an individual’s perception of attributes or similarity is an open question. However, as we demonstrated in Chapter 4, collaborative filtering techniques can be used to model individual aesthetic preferences; adapting these techniques for personalized font attributes is possible future work.

We have only scratched the surface of what can be done to improve font selection and make the difficult task of graphic design easier. Most immediately, it may be possible to combine the group and attribute interfaces into a single, more intuitive experience. We could also learn an “auto” button to automatically suggest a font for a design using a model trained from professional designs. Selecting fonts that are visually compatible (e.g., choose a header font to go with a specific body font) is also challenging; we could learn a joint model of compatible fonts from a corpus of designs. Finally, our set of attributes is limited; we could use natural language techniques to map any text to our existing set of attributes.
Conclusions

We conclude with a high-level discussion of important challenges and ‘lessons learned’ during this thesis, as well as thoughts on the future of computational design and aesthetics.

Modelling. We began this work by describing a simple linear model for color compatibility. While this model was useful for basic applications like improving a color theme, the model is limited in several ways. While we can analyze the weights, such analysis only provides clues about color compatibility. More research is required to develop a simpler model which can generalize to an arbitrary number of colors. Furthermore, the model does not include any knowledge of how colors are used in practice. For example, in graphic designs, certain colors may be used as accents, or have associations with particular objects. The model also cannot be used generatively to synthesize color themes in different styles.

By contrast, in Chapter 5, we described a energy-based layout model which was used to generate new layouts in a variety of styles. Our methodology involved studying design literature and examples for stylistic variation or functionality. We then created relevant energy terms, optimized designs with these terms, and then iteratively refined them. For example, our early experiments produced designs with a poor read order. These results let us to develop a flow heuristic to place more important elements higher in the design. Parameters were initially hand-tuned, but this became difficult as the model increased in complexity. Hand tuning 122 parameters to capture a single style is impractical, particularly when we wish to capture a wide range of styles. We therefore turned to NIO to estimate parameters from examples. NIO was also used to remove redundant energy terms; if after learning on several styles, an energy term was not used, it was removed.

This type of modelling is effective, but incredibly time-consuming; our model took approxi-
mately a year to develop. Furthermore, models of graphic design are particularly complex. Our model has dozens of energy terms, and still produces a limited range of styles. Important future work includes leveraging large datasets of designs to simply the modelling process. Given enough designs, it may be possible to transfer content between designs, similar to the Bricolage system [71], or learn layout templates.

**Aesthetic Features** In this work, we use an exploratory approach for feature creation and in Chapters 3, 5, 7, we create large feature sets for modeling color, layout, and typography. Exploratory feature creation is common in the aesthetic learning literature [61, 22, 108, 23]. In this approach, domain specific features are usually created based on literature such as photography books, common heuristics like the Rule of Thirds, or the researcher’s intuition. Features are then evaluated for their usefulness. Our process for creating features was similar, and in Chapter 3 we analyze the importance of different color features. Given the small number of designs available for layout modeling, analysis of features is not especially useful. However, with larger datasets, analysis of layout features such as symmetry and visual complexity will likely be valuable.

By contrast, recent works by Marchesotti et al. [90] and Karayev et al. [60] has shown that generic image features, such as SIFT, GIST, or features learned from deep neural networks, can perform better than handcrafted features. Furthermore, these approaches are simpler to implement and are more generally applicable than domain specific features. However, generic image features are less useful for analysis and interpretation. Finding simple and explainable models of design aesthetics is an important future goal. That said, the ease and effectiveness of generic image features encourages their use for tasks where interpretability is less important.

**Interface Development.** Building interfaces for designers is not a simple task. One theme of this work is the need for better tools, and particularly for automatic assistance in the design process. In practice however, such bold claims can be difficult to implement.

Adaptive interfaces, i.e. interfaces that automatically adjust elements, have great potential to allow design exploration more easily than traditional tools. For example, the interface of Chapter 6 automatically moved elements to avoid overlap, improve symmetry, and correct alignment, simplifying the interactions required for large-scale layout changes. However, building a successful adaptive interface means inferring the user’s goals, or making significantly better suggestions than the current design. In our experience, when our system inferred the desired goal and made the automatic update, it was quite satisfying. However, when the inferred goal was wrong, it produced a feeling of “fighting” the system. It is also hard to predict the results of optimizing
a complex energy-based model. In our system, the automatic updates were not always intuitive or consistent. Key challenges for building adaptive interfaces include determining the actions for which users will accept automatic updates, making intuitive changes to the layout, and clear signalling to the user for larger changes.

Design tasks produce a huge variance in responses, both in the creation of designs, as well as the evaluation. Evaluating interfaces is therefore difficult, and large sample sizes are required. Furthermore, it is not even clear that statistically measurable improvements in resulting designs are the main criteria by which to judge interfaces. A user’s enjoyment of the interface and design process is at least as important to consider. This issue was explored in Chapter 7, where the effect size of our attribute font interfaces was quite small, even though users enjoyed using attributes to select fonts.

Suggestion-based tools for creativity tasks should refrain from judging or critiquing a user’s design too strongly. As we discovered in Chapter 6, some people found that model-based suggestions were unnecessary, even detrimental, to their creative process. Early versions of our interactive layout tool performed worse, with more users reporting that the suggestions negatively impacted creativity. We incorporated that feedback and tried to encourage an exploratory approach to finding layouts, rather than one where user designs were being evaluated by the system and improved upon. To this end, refinements suggestions were call ‘Tweaks,’ instead of ‘Improvements.’ Style suggestions were labelled ‘Brainstorming,’ to emphasize they were meant as starting points for the design process, not the final result.

Crowdsourcing Challenges. In this work, we used crowdsourcing heavily for a variety of tasks, including perceptual studies of color preferences, A/B evaluations of synthesized designs, and testing new design interfaces. In general, we found crowdsourcing to be an invaluable technique for quickly and cheaply running experiments. However, there are challenges when using crowdsourcing for aesthetic tasks.

Responses to aesthetic tasks are usually subjective and difficult to validate. For most of our tasks, we included duplicate entries to catch workers who would quickly complete the tasks, but this approach is limited; it is still possible to do a poor job while noticing duplicates. For example, in our color study, one user rated thousands of color themes with three stars, far more than any other rating. The user was not malicious, and they did provide other ratings; they merely had an extremely low variance on their responses, so it is difficult to reject or block them from completing more tasks. Another problem is well-intentioned users who simply do not understand the instructions. However, users who expend minimal effort on aesthetic tasks are
more difficult to filter out.

A large motivation of MTurk workers is financial, so tasks should be reasonably priced. However, prior research by Mason and Watts [95], and our own experience, has shown that the task payment affects how quickly the task will complete, not the quality of results. Furthermore, aesthetic tasks like layout creation, or rating color themes, are more creative and fun than many other tasks on MTurk, so users will generally complete these tasks for less financial reward. However, a fun and well-paying task may in fact be too attractive, with a few users completing all the tasks, skewing later data analysis and learning.

In our experience, the majority of users on MTurk perform tasks reasonably. However, because of the difficulty of validating responses, we generally avoided putting large numbers of tasks online at any one time. When we did put large numbers of tasks online, such as the font attribute study, the prices tended to be lower to minimize overall study costs, but this also de-incentivized a small number of users to complete all our tasks. As another method to promote higher quality work, we also used a bonus structure for our layout tasks in Chapter 5. While we did not compare results with and without bonuses, we did receive positive feedback from users about the payment structure.

Separating Style and Content. One difficult problem with design modelling is the separation of style and content. For many design tasks, we would like to explore different styles without changing the content. For example, a system might suggest variations on a graphic, font, or layout. However, our perception of design content is often linked to its aesthetics. For a graphic, this is intuitive; a graphic of a hamburger is incongruous with a bookstore poster for example. But more subtly, if an important element, like a title, is placed so as to appear unimportant, there is a disconnect between the content and how we perceive it. Unsurprisingly, in our studies, we found that worst-rated layouts or font choices were generally those which were the hardest to visually parse and understand. These poor designs included illegible font choices, or randomly arranged layouts, making the design difficult to read. This result echoes the findings of Reber et al. [127], which suggest that aesthetics is heavily influenced by an object’s fluency, i.e., the ease of visual processing.

Few researchers have tackled this issue directly. Tenenbaum and Freeman [153] describe a method for separating style and content for fonts. An observation vector $y$ is decomposed into vectors for content $a$ and style $b$, which interact through a bilinear mapping $W$ (i.e., $y = a^T W b$). While this linear decomposition produces poor generative results, learning style parameters given examples is an important idea, and similar in spirit to our approach of learning styles
from multiple examples using NIO.

One way we approached this problem was by defining metadata for elements, such as importance or groups, along with relevant energy terms. These terms help enforce the content of the design, while other terms are more related to aesthetic style, such as symmetry. However, there is no clear-cut distinction in our model. While symmetry is primarily aesthetic, it does play a strong role in how we view designs, and symmetrical designs are often simpler to visually process.

To further investigate how content affects aesthetics, we ran a pilot MTurk study where users created layouts where the content was either rendered normally, or as an abstract coloured rectangle. We then compared how the layouts differ between abstract designs, and those with content. Figure 8.1 shows several examples. In general, there are many similarities between the two sets of layouts. Many layouts exhibit strong symmetry, and many elements have similar sizes, and are often placed equidistantly. Furthermore, both sets include many simple one or two column layouts. However, understanding content is key for creating more complex layouts, where relative sizes and spacing can vary, elements may overlap, and element types may have priors (e.g., actor’s names on a movie poster).

Even if we can build models of purely aesthetic style, such as the color compatibility models we investigated, applying these models to practical designs requires understanding the content. Better models of content might include more metadata from the users (e.g., by labelling elements as titles or body text), or by inference, such as detecting dates or locations in text blocks. Building versatile models of design content, which are not onerous for users to specify, is an important area of research.

**Design as Search.** As a research area, suggestion-based design tools are still in their infancy. It is therefore worth considering the more mature, but closely related, field of search interfaces, such as Google. Many aspects of graphic design are search processes. For example, in Chapters 6 and 7, we built prototype interfaces which allowed users to search possible layouts, as well as font choices. Other search tasks include choosing colors for backgrounds and fonts, or looking for relevant assets, such as clip art or background textures.

We can draw several lessons from successful search interfaces. First, search interfaces should be simple, but use sophisticated inference. Google presents users with a single text window, but tries to parse and understand the user’s query (e.g., auto-corrected spelling). By contrast, layout creation tools like Illustrator are extremely complicated, and try to infer very little about the user’s overall goals. For example, even the simple goal of finding a background image is
Figure 8.1: Abstract design layouts. In separate tasks, MTurk users created layouts for abstracted designs, as well as designs with visible content. Many abstracted designs are reasonable for unknown content, particularly for simple templates like one or two columns. However, templates do not generalize to more complex layouts, where content significantly affects the layout.

incredibly time-consuming, and involves many repeated steps: searching online, saving and opening files, setting the background in the design tool, possibly resizing, etc. Second, design interfaces should model the user as much as possible. Search engines uses user location, search history, device, among other features, to help guide its search recommendations. Similarly, if you know a user’s preference for font or layout styles, you should recommend relevant design suggestions. Third, interfaces should avoid showing users too many results (e.g., like current
font selection interfaces). Most search engines commonly return 10 results on the first page, and the desired page is usually within this first set.

Finally, search interfaces use metrics to evaluate success, such as number of queries in a session, or long-term user retention [64]. To build a successful design interface, we should consider possible metrics, including time spent, number of interactions, number of alternatives explored, user satisfaction with the design, final quality of the design, etc. As in search, these metrics may potentially conflict [64]; in Chapter 6, we found the counterintuitive result that our new suggestion-based interfaces produced better designs, but were not preferred to the baseline without suggestions, which was perceived to allow more creativity. However, defining metrics is a key part of evaluating data-driven tools, and a powerful approach for guiding interface design.

**Future Work.** The most important future work is integrating different design components into a holistic interface which provides suggestions for colors, font, layouts, image processing operations, alternative assets, etc. Such an interface should better understand the design process. For example, a design interface could present alternatives to the users, allow simple and intuitive exploration of the design space, incorporate collaboration and feedback from other users, and analysis of how people will perceive the design (e.g., can people read this text at a small resolution?)

While we used a large design dataset to train the importance model of Chapter 5, we did not learn a layout model from this data. Unfortunately, it is difficult to learn layouts from rasterized designs. Improved parsing is one option, but with the growth of web-based design sites like Canva, datasets of vector-based designs may also appear within the next few years. Design datasets will be invaluable in many ways, but one important goal is to learn a higher-level representation of design. For example, extracting common templates could improve layout suggestions. Another possibility is training grammar-based models like Talton et al. [149]. These datasets may also allow automatic feature extraction using recent advances in deep learning [48].

There is significant remaining work in understanding typography. Large design datasets could be used to train models for suggesting font choices given a particular design. Similarly, a joint font compatibility model for predicting how well fonts go together would also be useful. Synthesizing fonts that match specific attributes, or that fit a particular design, is another possible research area.

Another important area of future work is developing more mid-level vision algorithms to use within design tools. In this thesis, we developed a few such algorithms, such as predicting visual importance and estimating font attributes, to enable our systems. However, there remain many
unsolved vision problems which would be useful for building design tools, including estimating aesthetic ratings for designs, perceptual grouping of elements, a distance metric for designs, view order, or design attributes.

In this thesis, we have made a few steps towards building a computational model of design. Our models for color, layout, and typography, along with relevant applications and interfaces have demonstrated the power of machine learning approaches in this domain. Building tools to help people with the difficult task of design is an inspiring challenge. Not only must we examine deep and fascinating questions of perception, but there is the hope of helping millions of people to create designs of purpose and beauty.
Bibliography


[120] Stephen Palmer, Yurika Hara, and William Griscom. Why the "Rule of Thirds" is Wrong.


Appendix A: Color Theme Dataset Details

For Kuler themes, the mean rating is \( \bar{r} = 3.14 \) with a variance \( \sigma^2 = 0.52 \). The mean variance \( \bar{\sigma}^2 \), indicating the typical disagreement of users on themes, is 1.31. These statistics are defined as follows. For theme \( t_i \) with \( N \) user ratings \( r_{i,j} \), the theme mean rating is \( r_i = \frac{1}{N} \sum_j r_{i,j} \). The mean rating over \( M \) themes is \( \bar{r} = \frac{1}{M} \sum_i r_i \) with variance \( \sigma^2 = \frac{1}{M} \sum_i (r_i - \bar{r})^2 \). The mean variance \( \bar{\sigma}^2 = \frac{1}{M} \sum_i \frac{1}{N} \sum_j (r_{i,j} - r_i)^2 \). The mean number of ratings per theme is 5.77, median 3 and max 1440. The distribution of ratings is (17%,11%,25%,24%,24%) for 1-5 stars.

For COLOURLovers themes, we define a numerical rating for each theme as: \( r(h,v) = (h - \bar{h}(v))/\sigma(v) + 3 \), where \( \bar{h}(v) \) and \( \sigma(v) \) are linear functions corresponding to the mean and standard deviation of hearts for a given number of views. Specifically, \( \bar{h}(v) = av + b \). The parameters \( a \) and \( b \) are fit as follows. We discretize the number of views \( v \) into bin \( i \), and define \( m_i \) be the mean hearts per theme with \( n_i \) views. The parameters are set by minimizing \( L_1 \) error:

\[
\sum_i |(av_i + b) - m_i|,
\]

yielding values of \( a = 0.0152 \) and \( b = -0.263 \). Similarly, \( \sigma(v) \) is fit to the variances, giving \( a = 0.0128 \) and \( b = 0.218 \). See Fig. 3.2 for a plot. The mean of the estimated ratings \( r(h,v) \) over all themes is \( \bar{r} = 2.98 \) and variance \( \sigma^2 = 0.75 \).

MTurk tasks required each participant to rate 30 themes on a discrete scale of 1 to 5. Themes were shown on black to match the Kuler interface. Theme names were not shown. Each task was worth $0.02. Each theme was rated by 40 participants, with 1,301 participants total. Workers were required to indicate their gender, age, and the country they have lived longest in. Participants were evaluated for consistency by including duplicate themes, with 263 workers removed. Inconsistency was measured with the standard deviation for each pair of duplicate ratings (\( \sigma_d \)). The average of all duplicates (\( \bar{\sigma}_d \)) was found, and if \( \bar{\sigma}_d > 0.7 \), the user was removed. If the standard deviation of all ratings was less than 0.6, the user was removed for using too few rating values. The mean number of ratings per theme is 39.3, median 35, and max 80. The mean rating is \( \bar{r} = 2.98 \) with a low variance of \( \sigma^2 = 0.11 \) since theme ratings are aggregated over many user ratings. The mean variance \( \bar{\sigma^2} = 1.14 \). Since MTurk themes are sampled from Kuler, we can also directly compare the Kuler statistics for those 10,743 themes. Of that Kuler subset, the mean rating \( \bar{r} = 3.19 \), variance \( \sigma^2 = 0.40 \), and mean variance \( \bar{\sigma^2} = 1.64 \). This indicates that for the same themes, MTurk users generally give lower scores with less disagreement. The distribution of MTurk ratings is (10%, 22%, 35%, 24%, 9%) for 1-5 star ratings. Note in Kuler 24% of ratings were 5 stars with only 9% in MTurk.
Figure 8.2: Theme-template distance example. The theme is defined by the 5 HSV values (8,80,18), (16,80,54), (164,85,23), (75,7,96), (164,36,63). We remove the 4th color since it is lower than the saturation threshold (15) and sort the remaining hues: (164,164,16,8). The hue differences (with wrap-around for the final color) are then (0,148,8,204), which are sorted to give (204,148,8,0) and padded with zeros for the removed color. Intuitively, the two large numbers indicate two widely separated hue clusters. As expected, the sum of absolute hue difference from the complementary template is 64, which is much smaller than the triad template distance of 224.

Appendix B: Theme-Template Distance

We compute the distance between a theme and a template as follows. For a theme, we first sort by hue and then compute the hue differences for saturated and light colors, with unsaturated or dark colors given a hue difference of 0. We use a threshold of 15 for both saturation and lightness.

For example, a theme with only 2 exactly complementary hues has a hue difference vector of (180,180,0,0,0). For a theme with 3 triadic hues, the difference vector would be (120,120,120,0,0). These vectors are then compared to exemplars for each template. The vectors above are the exemplars for the I and R templates respectively. The distance between a theme and template is the sum of absolute differences. Note that for symmetric hue templates (all except C and L), the hue difference vectors and exemplars can be sorted. C and L require multiple exemplars. For a concrete example of the theme to hue template distance, please refer to Fig. 8.2.

Templates in Matsuda’s model [96] are defined as sectors over the hue wheel instead of fixed angular differences. For our exemplars, we use the centers of the template sectors, or equally spaced hues in the sectors, to calculate the distance. Directly using hue sectors has several problems. For example, simpler templates are contained within larger ones; a monochromatic theme would have a distance of 0 to all templates. It is also difficult with sectors to evaluate slight differences from the geometric ideal of classical templates.
Appendix C: Optimization Proposals

Simulated Annealing Proposals. The optimization of Chapter 5 uses several different proposals to deal with the complexity of our function and dependencies between elements:

- **Update Single Element Position.** For updating the position, a normally distributed offset is added to the current position ($\sigma_{loc} = 0.1$), elements are moved along an axis to fixed positions, or elements moved to an empty part of the design.

- **Update Height.** Element heights are updated by adding a normally distributed offset to the current height ($\sigma_{hei} = 0.2$).

- **Align Elements.** Element align with another on a single axis, either with a single alignment type, or on all three (i.e., by changing the height or width as well).

- **Swap Two Elements.** The position of two elements are swapped.

- **Update Element Group.** If the user specifies an element group, height and position changes are proposed for the entire group, since these heights and positions are correlated in the energy function.

- **Switch Alternate.** If an element has alternates, this proposal will randomly switch to one of the alternates.

- **Update Alignment Group.** If the alignment labeling has detected an alignment group, the entire group’s position is shifted by a normally distributed offset ($\sigma_{aloc} = 0.05$).

- **Reduce Alignment Error.** Given a detected alignment, the current misalignment error is reduced. The direction of minimum error is simply $\mathbf{x} = \mathbf{C}\mathbf{b}$ where is $\mathbf{C}$ is an $2n \times 2n$ matrix indicating if elements $i$ and $j$ are aligned along the $x$ and $y$ axes, and $\mathbf{b}$ is the alignment difference. $n$ is the number of elements. A line-search is performed along $\mathbf{x}$ to choose the state with lowest energy.

- **Fill Image.** Some designs have very large graphics, one proposal scales a graphical element to match the design height or width.

Proposals types are all equally likely to be selected except for the **Fill Image** proposal, which is proposed less frequently (20%) due to its low acceptance rate.
Parallel Tempering Proposals. In Chapter 6, the parallel tempering algorithm uses the following proposals when sampling:

- **Update Element Position.** For updating the position, a normally distributed offset is added to the current position ($\sigma_{loc} = 0.1$), elements are moved along an axis to fixed positions, or elements moved to an empty part of the design.

- **Update Element Height.** Element heights are updated by adding a normally distributed offset to the current height ($\sigma_{hei} = 0.05$).

- **Align Two Elements.** Two elements are aligned on a single axis and alignment type.

- **Swap Two Elements.** The position of two elements are swapped.

- **Switch Alignment.** If a text element has more than one line, this proposal will randomly switch the alignment.

- **Switch Alternate.** This proposal will switch the element to an alternates (i.e, another version of the text with different line breaks), if such alternates exist.

- **Update Aligned Elements.** If an element is aligned with other elements, then the entire set of elements is shifted by a normally distributed offset ($\sigma_{aloc} = 0.1$).

- **Scale Types.** The heights for all text or graphical elements are updated by scaling by a normally distributed amount ($\sigma_{hei} = 0.1$).

Proposals types are all equally likely to be selected except for the **Switch Alignment**, **Swap Two Elements**, and **Scale Types** proposals, which are proposed less frequently (50%), as well as the **Update Element Position** proposal, which is proposed twice as often (200%). The frequencies were set experimentally.
Appendix D: Font Features

*Size and Area.* For both the lower and upper case ‘X’, we measure the width, height, and width/height ratio. We find each character's area, and the ratio of the area with the bounding box. We then find the min/max/mean over all characters. We also compute the height of the biggest ascender or descender in the character set: the length that extends above/below the mean line of the font.

*Spacing.* We measure the horizontal and vertical spacing between upper and lower case ‘X’. That is, for the horizontal spacing, we measure the space between the two characters ‘xx’ and ‘XX’. We also compute vertical spacing for characters with descenders.

*Outlines.* For each character, we compute the sum of the outline arc lengths, as well as the minimum and maximum of the set. We then compute the min/max/mean for all characters. We also find the mean number of independent curves for each character.

*Curvature Histograms.* We compute curvature histograms for curved and non-curved characters. The curvature points are computed by iterating over the character, and computing the angle between adjacent points. We use 10-bin histograms, along with histogram entropies. We then compute the Earth Mover's Distance between lower and upper case curvature histograms. For each character, we also find the max and mean curvature, and the entropy of the curvature histograms, then compute the min/max/mean over all characters.

*Orientation and Width.* To measure the orientation of the characters, such as italic or slanted fonts, we create a point set from the character ‘L’, compute PCA on the points, then find the angles and magnitudes of the first two principal components. We estimate the stroke width by taking the horizontal width of ‘L’, ‘I’, and the ‘i’. We also estimate the stroke width by the character ‘O’, both along the x-axis and the y-axis, as well as the ratio of both widths.
Appendix E: Font MTurk Study Details

Attribute Study. Each Human Intelligence Task (HIT) on MTurk consists of 16 comparison tasks, along with four control tasks for quality control. Two of the four control tasks check for consistency; we repeat two of the 16 tasks with the order of the fonts swapped. The other two control for correctness; we add two font pairs with the attribute “thin” which are unambiguous and should have a clear answer. We discard any HITs in which the worker fails two or more control questions; the final rejection rate was 8.1%. Workers were paid $0.07 per HIT.

For each attribute, the total number of comparison tasks is \( mn \), where \( m \) is the number of fonts, and \( n \) is the number of pairwise comparisons per font. The division by 2 appears since two fonts appear in each comparison. For our dataset, \( m = 200 \) and \( n = 8 \), providing 800 comparison tasks for each attribute, with font pairs selected randomly. Each comparison task was completed by 8 unique workers, providing \( 8n = 64 \) individual responses for each font/attribute pair. Over all attributes and fonts, this produces a final dataset of 198,400 individual responses for 639 unique workers.

Selecting the number of pairwise comparisons \( n \) is an important choice when estimating font attributes. If \( n \) is too low, the estimated attributes are inaccurate. Exhaustive testing of all pairs (i.e., \( n = 200 \)) is cost-prohibitive and unnecessary since the attribute values will converge to accurate values after far fewer comparisons. To determine a reasonable \( n \), we ran a smaller study on 5 attributes varying \( n \) from 2 to 15. For each \( n \), we evaluated the mean log-likelihood of a testing set of 5 additional comparisons per font. In Figure 8.3, we show a figure plotting the log-likelihood for \( n \) for each attribute. We found the log-likelihood plateaued after \( n = 8 \), so we used this value in our final study for the remaining attributes.

Similarity Study. To obtain data to train the model, we conduct a crowdsourced study focused on font similarity. Workers are presented with a reference font A and two fonts (B and C) and are asked to decide whether B or C is more similar to A than the other. See an example of one such task in Figure 7.8. Triplets were randomly sampled from the previously described 200 font training set.

Each HIT contains 16 such tasks, as well as 4 control questions. For consistency, we repeat two tasks by swapping the fonts B and C. For correctness, we include tasks where one of B and C is the same font as A; the user is expected to recognize that a font is more similar to itself than any other font. We reject HITs that fail at least two of the four control questions. We also ignore users if 20% or more of their HITs were rejected, leading to a final rejection rate of 9.11%.
Figure 8.3: Mean log-likelihood values of the objective function with varying numbers of pairings per font. For the final study on all 31 attributes, we used 8 pairings per font.

Workers are paid $0.07 per HIT. To obtain multiple opinions per triplet we created 130 HITs with at most 15 workers per HIT. After rejection, the average number of users per triplet was 13.6. The total dataset has 2,340 triplets and 35,387 individual comparisons. Every font in the training set was in at least 21, and at most 57, triplets.

**Target Study.** Each HIT contained 5 fonts to match, with 10 HITs total. The same sequence of target fonts was also used for each interface. Each interface is used by 15 workers, giving 750 font selections for each interface. Workers were paid $0.50 per HIT. Bonuses were also promised to users for the nearest font selections, with the top 25% of users receiving a bonus ranging from $0.10 to $0.50.

**Design Task Study.** For each interface (attribute, group, and baseline), we asked 30 workers to choose fonts for 15 designs, giving 1,350 designs total. Each HIT contained 5 designs and used a single interface. Workers were paid $0.50, and bonuses were also promised to users for the best designs, as evaluated by other users. The top 25% of users received a bonus ranging from $0.10 to $0.50. 200 workers completed the HITs.

We next evaluated the designs created using the new interfaces against the designs created
by the baseline interface. 2AFC testing was performed with designs selected from the baseline interface and either the attribute or group interface; each selected design was compared to 10 other designs. Each HIT contained 18 comparisons, with 2 duplicates added for consistency. 8 users performed each HIT and were paid $0.07. 455 HITs were created, producing 62,766 individual comparisons.