Visualizing Multi-level Structures in Data

by

Zipeng Liu

Bachelor of Science, Peking University, 2015

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES

(Computer Science)

The University of British Columbia
(Vancouver)

August 2021

© Zipeng Liu, 2021
The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

**Visualizing Multi-level Structures in Data**

submitted by Zipeng Liu in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

**Supervisory Committee:**

Tamara Munzner, Computer Science  
*Supervisor*

Giuseppe Carenini, Computer Science  
*Supervisory Committee Member*

Michiel van de Panne, Computer Science  
*Supervisory Committee Member*

**Examine Committee Members:**

William Evans, Computer Science  
*University Examiner*

Sathish Gopalakrishnan, Electrical and Computer Engineering  
*University Examiner*

Enrico Bertini, Computer Science and Engineering, New York University Tandon School of Engineering  
*External Examiner*
Abstract

Visualization is an important tool to analyze data, but there emerge various challenges from complex analytical data and tasks. In this dissertation, I present four projects that were motivated by these challenges, situated in the nested model proposed by Munzner, which consists of four layers to describe the components in visualization: domain, data and task, encoding design, and algorithm.

In **ADVIEW**, to address the challenges of comparing many phylogenetic trees in the domain of biology, I propose a visual encoding to compress a tree representation, design and implement a multi-view interactive tool to handle the multiple levels of detail in a tree collection dataset, ranging from the whole collection, through subsets of trees, individual trees, subtrees, to leaf nodes. In **SPRAWLTER**, to address the existing visual encoding problems of readability metrics for node-link graphs, I propose two novel metrics to measure a finer-grained clutter and to balance the geometric sparseness and clutter. These metrics recognize different levels of visual saliency such as metanodes and leaf nodes in multi-level graphs. In **LOGSEG**, to fulfill user demands for chunking actions in the domain of image editing software, I propose a segmentation model for the action logs to serve the demands that require different chunking granularities. For example, smart undo for going back to a previous user task needs a low-level chunking, while managing an overview of milestones needs a high-level one. In **CORGLIE**, to fill the gap in visual qualitative evaluation of graph neural networks (GNNs) in the domain of machine learning, I propose an approach and design a tool to explore correspondences between a graph and its embedding to check how different levels of structures are preserved from
the input graph to the output embedding. I also design a new graph layout to reveal how a GNN leverages node neighbors and computes an embedding.

I identify a common theme among these projects: multi-level structures. They consist of nesting subsets of data points that are relevant to the analytical tasks. I demonstrate how to exploit them in the visualization if provided in hierarchical data, or to compute them for non-hierarchical data.
Lay Summary

Data acquired in many domains in our society today needs to be analyzed by users with visualization, which produces graphical representations from the data and allows the users to interact with the graphics. For different analytical data and tasks in different domains, there are different challenges to visualize the relevant information. I propose to focus on the visualization of multi-level structures, the internal structures in data which consists of multiple levels, to address some of these challenges. I introduce four projects to demonstrate how to exploit multi-level structures from data with hierarchical information, and how to compute them from data without hierarchical information, in three different domains including biology, software usability, and machine learning.
Preface

Parts of this dissertation have appeared in either peer-reviewed publications or ongoing journal submissions.

Chapter 2 is based on the following published journal paper, code-named ADVIEW.


Chapter 3 is based on the following published journal paper, code-named SPRAWLTER.


Chapter 4 is based on the following published journal paper, code-named LOGSEG.

Chapter 5 is based on the following recent journal submission, code-named CORGIE.


As the first author of all four papers above, I took a leading role in the research projects and contributed in many aspects. I characterized the domain discipline of the project: phylogenetics in ADVIEW, image editing software in LOGSEG, neural networks for graph machine learning in CORGIE, which involved meeting and interviewing with domain experts and stakeholders, reading and summarizing literature, learning to perform the analytical task myself (e.g. using Photoshop, training graph neural networks), etc. In this process, I also abstracted the data and tasks from domain-specific language to general visualization language. I took charge in most of the visual design work, and I am the sole developer to implement them with web-based tools (HTML, CSS, Javascript). I planned and executed the evaluation of the proposed tools or algorithms, such as conducting user study sessions with target users and measuring performances on a computer. I am the main writer of the manuscripts, which involves a few composing and editing passes before submission, and addressing issues raised by reviewers. I also wrote most of the supplemental documents for these papers, and filmed the demonstration videos for the proposed tools.

My supervisor Tamara Munzner shepherded these projects by providing both high-level guidance and concrete feedback throughout all project life cycles. During the writing phases, she ensured a proper paper structure and also edited my drafts in great detail. In ADVIEW, Shing Hei Zhan is the phylogenetic expert who provided information about the domain, feedback to my visual design and the tool. The other coauthors are visualization researchers who have provided help in various phases in the projects like applying for funding, connecting to stakeholders, reviewing literature, designing interfaces, recruiting participants for user studies, editing the manuscripts, and more.
All user studies in this dissertation have been approved by the UBC Behavioural Research Ethics Board (BREB) with the certificate No. H10-03336.
# Table of Contents

Abstract ................................................................. iii

Lay Summary ........................................................... v

Preface ................................................................. vi

Table of Contents ....................................................... ix

List of Tables ........................................................ xvi

List of Figures ........................................................ xvii

Acknowledgments ....................................................... xxii

1 Introduction ........................................................... 1
   1.1 Four layers of visualization concerns ......................... 2
   1.2 Challenges ...................................................... 3
   1.3 Multi-level structures ........................................... 5
   1.4 Contributions ................................................... 6
   1.5 Evaluation methodology ....................................... 8

2 ADView: Aggregated Dendrograms for Visual Comparison Between Many Phylogenetic Trees .................................................. 10
   2.1 Motivation ....................................................... 11
   2.2 Related work ................................................... 13
3 Sprawlter: Combining Sprawl and Area-aware Clutter for Graph Readability Metric ............................... 55

3.1 Motivation ........................................ 56

3.2 Related work .................................... 59

3.2.1 Graph drawing readability metrics .......... 59

3.2.2 Evaluation of graph drawing metrics ........ 62

3.3 sprawlter metrics .................................. 62

3.3.1 Graph layout definitions ...................... 63

3.3.2 Measuring geometric overlap .................. 63

3.3.3 Mapping measurements to penalties .......... 64

3.3.4 Summing penalties .............................. 65

3.3.5 Combining clutter metrics with sprawl ....... 66

3.4 Penalty mapping function ......................... 68

3.4.1 General requirements .......................... 68

3.4.2 Specific choices ................................. 70

3.4.3 Formal function instantiations ................. 71

3.4.4 Parameter space analysis .................... 73

3.5 Data generation .................................... 74

3.6 Results .......................................... 75

3.6.1 Analysis approach ............................. 76

3.6.2 Comparative analysis ........................... 77

3.6.3 Computational time ............................. 84

3.7 Discussion and Future Work ....................... 85

3.8 Summary .......................................... 87

4 LogSeg: Data-driven Multi-level Segmentation of Image Editing Logs 88

4.1 Motivation ........................................ 89

4.2 Related Work .................................... 92

4.2.1 Event Sequence Visualization ................ 92

4.2.2 Workflows and Graphical Histories .......... 92

4.2.3 Event Log Models ............................. 93

4.3 Approach ........................................ 95

4.3.1 Data Generation ............................... 95
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Motivation</td>
<td>120</td>
</tr>
<tr>
<td>5.2 Background</td>
<td>123</td>
</tr>
<tr>
<td>5.2.1 Graph neural networks</td>
<td>123</td>
</tr>
<tr>
<td>5.2.2 Motivating scenarios</td>
<td>124</td>
</tr>
<tr>
<td>5.3 Data and task abstraction</td>
<td>126</td>
</tr>
<tr>
<td>5.3.1 Overview</td>
<td>127</td>
</tr>
<tr>
<td>5.3.2 Data Space Abstraction</td>
<td>128</td>
</tr>
<tr>
<td>5.3.3 Task abstraction</td>
<td>131</td>
</tr>
<tr>
<td>5.4 CorGIE design</td>
<td>132</td>
</tr>
<tr>
<td>5.4.1 Overview</td>
<td>133</td>
</tr>
<tr>
<td>5.4.2 View design</td>
<td>134</td>
</tr>
<tr>
<td>5.4.3 View coordination</td>
<td>139</td>
</tr>
<tr>
<td>5.4.4 Implementation</td>
<td>140</td>
</tr>
<tr>
<td>5.4.5 Design alternatives</td>
<td>141</td>
</tr>
<tr>
<td>5.5 K-hop layout</td>
<td>141</td>
</tr>
<tr>
<td>5.5.1 Algorithm</td>
<td>142</td>
</tr>
<tr>
<td>5.5.2 Scalability</td>
<td>144</td>
</tr>
<tr>
<td>5.5.3 Design alternatives</td>
<td>145</td>
</tr>
<tr>
<td>5.6 Results</td>
<td>146</td>
</tr>
<tr>
<td>5.6.1 Usage scenario 1: Movie Recommendation</td>
<td>147</td>
</tr>
<tr>
<td>5.6.2 Usage scenario 2: Cora</td>
<td>150</td>
</tr>
<tr>
<td>5.6.3 Expert study overview</td>
<td>154</td>
</tr>
<tr>
<td>5.6.4 Expert session 1: Cora decision boundary</td>
<td>154</td>
</tr>
<tr>
<td>5.6.5 Expert session P2: USAir</td>
<td>156</td>
</tr>
<tr>
<td>5.6.6 Expert feedback summary</td>
<td>158</td>
</tr>
<tr>
<td>5.7 Related work</td>
<td>159</td>
</tr>
<tr>
<td>5.7.1 Visualizations for machine learning models</td>
<td>159</td>
</tr>
<tr>
<td>5.7.2 Latent space interpreter</td>
<td>160</td>
</tr>
<tr>
<td>5.7.3 GNN interpreter</td>
<td>161</td>
</tr>
<tr>
<td>5.7.4 Graph layout</td>
<td>162</td>
</tr>
<tr>
<td>5.8 Discussion and future work</td>
<td>162</td>
</tr>
<tr>
<td>5.9 Summary</td>
<td>163</td>
</tr>
</tbody>
</table>
# Reflection and Conclusions

6.1 Limitations .................................. 165
6.2 Reflection on multi-level structures ............... 166
6.3 Reflection on collaboration ........................ 168
6.4 Reflection on research topic choices .............. 169
6.5 Concluding remarks .............................. 170

## Bibliography

A Supplemental Material for ADView ........................... 192
A.1 View coordination details .......................... 192
A.2 Cluster AD gradient coloring ....................... 194
A.3 Algorithm details ................................ 196
  A.3.1 AD layout function parameters ................ 196
  A.3.2 Front-end caching for rendering ADs .......... 196
A.4 Expert Interview Study ............................. 199
  A.4.1 Participants ...................................... 199
  A.4.2 Interview Questions ............................... 199
A.5 Full Screenshots of Usage Scenario 1: 1KP pilot study .... 202
  A.5.1 Sister group of land plants ..................... 202
  A.5.2 Early diversification of land plants ........... 207
A.6 Usage scenario 2: TreeFam .......................... 209
  A.6.1 PROTOSTOMIA and DEUTEROSTOMIA ........... 210
  A.6.2 ECYSOZOA and LOPHOTROZOA ................... 215
A.7 Screenshots of Information Density Comparison .......... 218
A.8 Case Studies ...................................... 220
  A.8.1 Full Screenshots of Case Study 1 ............... 220
  A.8.2 Full Screenshots of Case Study 2 ............... 220
  A.8.3 Case Study 3 ...................................... 222
A.9 Screenshots of Dataset Upload ........................ 224

B Supplemental Material for Sprawlter ...................... 228
B.1 Parameter analysis ................................ 228
  B.1.1 Minimum penalty fraction ....................... 229
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1.2</td>
<td>EE curve shape</td>
<td>233</td>
</tr>
<tr>
<td>B.2</td>
<td>Pseudo-code for NE and EE area-aware metric</td>
<td>235</td>
</tr>
<tr>
<td>B.3</td>
<td>List of graph layouts</td>
<td>237</td>
</tr>
<tr>
<td>B.4</td>
<td>Computational time</td>
<td>238</td>
</tr>
<tr>
<td>B.5</td>
<td>Computational pipeline</td>
<td>240</td>
</tr>
<tr>
<td>C</td>
<td>Supplemental Material for LogSeg</td>
<td>242</td>
</tr>
<tr>
<td>C.1</td>
<td>Features</td>
<td>242</td>
</tr>
<tr>
<td>C.1.1</td>
<td>Commands</td>
<td>242</td>
</tr>
<tr>
<td>C.1.2</td>
<td>Layer</td>
<td>247</td>
</tr>
<tr>
<td>C.2</td>
<td>Data Collection</td>
<td>248</td>
</tr>
<tr>
<td>C.3</td>
<td>Hyperparameter Tuning for SVM</td>
<td>252</td>
</tr>
<tr>
<td>C.3.1</td>
<td>Window size $k$</td>
<td>252</td>
</tr>
<tr>
<td>C.3.2</td>
<td>Threshold $t$</td>
<td>253</td>
</tr>
<tr>
<td>C.4</td>
<td>Results</td>
<td>255</td>
</tr>
<tr>
<td>C.4.1</td>
<td>Labeling interface</td>
<td>255</td>
</tr>
<tr>
<td>C.4.2</td>
<td>Visualization of sessions in test set</td>
<td>257</td>
</tr>
<tr>
<td>C.5</td>
<td>Rule-based model</td>
<td>266</td>
</tr>
<tr>
<td>D</td>
<td>Supplemental Material for CorGIE</td>
<td>268</td>
</tr>
<tr>
<td>D.1</td>
<td>Concrete task examples</td>
<td>268</td>
</tr>
<tr>
<td>D.2</td>
<td>Configurable settings, filters and menus</td>
<td>269</td>
</tr>
<tr>
<td>D.3</td>
<td>Alternative visual design for topology space</td>
<td>269</td>
</tr>
<tr>
<td>D.4</td>
<td>Results</td>
<td>275</td>
</tr>
<tr>
<td>D.4.1</td>
<td>Usage scenario 1: Movie Recommendation</td>
<td>275</td>
</tr>
<tr>
<td>D.4.2</td>
<td>Usage scenario 2: Cora</td>
<td>280</td>
</tr>
<tr>
<td>D.4.3</td>
<td>Expert session P1: Cora decision boundary</td>
<td>283</td>
</tr>
<tr>
<td>D.4.4</td>
<td>Expert session P2: USAir</td>
<td>285</td>
</tr>
<tr>
<td>D.5</td>
<td>Semi-structured interview in user study</td>
<td>288</td>
</tr>
</tbody>
</table>
List of Tables

Table 4.1 The five kinds of layer relationships ............................................. 101
Table 4.2 Corresponding coefficients to features in SVM model ................. 107
Table 4.3 The performance metrics of sessions in the three task types ....... 112
Table 5.1 Computational benchmarks for the K-hop layout ..................... 144
Table 6.1 A summary of contributions of my four projects covering the four layers in the nested model .................................................. 170
Table A.1 View coordination of visual encoding across all levels of detail and all views ................................................................. 193
Table A.2 List of participants and datasets used in the study .................. 199
Table B.1 Full list of graph layouts ............................................................... 237
Table B.2 Computation time of each layout for three metric families and two approaches ............................................................ 239
Table C.1 Summary of task type ................................................................. 251
List of Figures

Figure 1.1  Munzner’s nested model [124] showing four layers of visualization concerns ........................................ 2
Figure 1.2  Illustrations of multi-level structures ........................................ 5
Figure 1.3  Evaluation methods under Munzner’s nested model [124] .................. 8

Figure 2.1  The problem space of visual tree comparison for two scalability dimensions ........................................ 15
Figure 2.2  Illustrations for the tasks and corresponding branches .................. 20
Figure 2.3  Visual encoding of an aggregated dendrogram .......................... 24
Figure 2.4  Illustration of how the AD adapts to available space .................. 26
Figure 2.5  Two examples of clustering structurally similar aggregated dendrograms ........................................ 27
Figure 2.6  Five versions of the aggregated dendrogram design in chronological order of invention ........................................ 29
Figure 2.7  Screenshot of ADView comparing a reference tree against a tree collection ........................................ 29
Figure 2.8  Illustration of the Tree Distribution view ................................. 31
Figure 2.9  Pairwise comparison between the reference tree and a user-specified target tree from tree collection .................. 33
Figure 2.10 Legends for the visual encodings ........................................ 36
Figure 2.11 Annotated screenshots for a research question in the 1KP pilot study: which is the sister group of LP .................. 45
Figure 2.12 Detail views of P1’s findings ........................................ 49
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.13</td>
<td>Cropped figures of P2’s findings</td>
<td>50</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of approaches to measuring node-node crossing in three synthetic 2-level graph layouts</td>
<td>58</td>
</tr>
<tr>
<td>3.2</td>
<td>Increasing amount of overlap between node-node, node-edge, and edge-edge pairs</td>
<td>64</td>
</tr>
<tr>
<td>3.3</td>
<td>General requirements for area-aware penalty mapping function</td>
<td>69</td>
</tr>
<tr>
<td>3.4</td>
<td>Illustration of the penalty mapping function for node-node overlap</td>
<td>72</td>
</tr>
<tr>
<td>3.5</td>
<td>The NN penalty mapping function of the smallest node pair with different $\alpha$</td>
<td>74</td>
</tr>
<tr>
<td>3.6</td>
<td>Increasing overlap between red and blue metanodes in a small 2-level synthetic graph</td>
<td>78</td>
</tr>
<tr>
<td>3.7</td>
<td>Metrics for a single-level synthetic graph with variable node sizes</td>
<td>79</td>
</tr>
<tr>
<td>3.8</td>
<td>Metrics for large real-world graph layout by FM3 in Tulip</td>
<td>80</td>
</tr>
<tr>
<td>3.9</td>
<td>Metrics for two large multi-level real-world graph layouts by GrouseFlocks</td>
<td>81</td>
</tr>
<tr>
<td>3.10</td>
<td>Increasing overlap between blue metanodes and edges in a small 2-level synthetic graph</td>
<td>82</td>
</tr>
<tr>
<td>3.11</td>
<td>Metrics for two large multi-level real-world graph layouts by GrouseFlocks</td>
<td>83</td>
</tr>
<tr>
<td>3.12</td>
<td>Decreasing crossing angles between edges in a small single-level synthetic graph</td>
<td>83</td>
</tr>
<tr>
<td>4.1</td>
<td>Concrete example of session segmentation</td>
<td>90</td>
</tr>
<tr>
<td>4.2</td>
<td>Distributions of the five feature values of non-boundary and boundary events</td>
<td>107</td>
</tr>
<tr>
<td>4.3</td>
<td>Session visualizations</td>
<td>111</td>
</tr>
<tr>
<td>5.1</td>
<td>Motivation of CorGIE</td>
<td>121</td>
</tr>
<tr>
<td>5.2</td>
<td>Data and task abstraction</td>
<td>128</td>
</tr>
<tr>
<td>5.3</td>
<td>Full screenshot of CorGIE on the Movie dataset</td>
<td>133</td>
</tr>
<tr>
<td>5.4</td>
<td>K-hop layout algorithm in four steps</td>
<td>142</td>
</tr>
</tbody>
</table>
Figure 5.5  Design alternatives to K-hop layout  .................................. 145
Figure 5.6  Check overview of the Movie scenario  ................................. 148
Figure 5.7  Check movie recommendations ........................................... 149
Figure 5.8  Explore overview of Cora dataset ........................................... 152
Figure 5.9  Find problematic node pairs in the Cora dataset ................. 153
Figure 5.10  P1 explored the decision boundary in the Cora dataset ...... 155
Figure 5.11  The USAir dataset without focal nodes .............................. 156
Figure 5.12  Find buggy node pairs ................................................. 157
Figure 5.13  Screenshot of P2 focusing on a buggy node pair ............... 157

Figure A.1  A duplicate screenshot of ADView for reference of the view coordination table ................................................................. 193
Figure A.2  Illustration of gradient coloring in a cluster AD .................... 194
Figure A.3  Illustration of different encoding representation for the same normal distribution ...................................................... 195
Figure A.4  Illustration of parameters in an aggregated dendrogram layout 197
Figure A.5  Pipeline on the frontend to render the aggregated dendrograms 198
Figure A.6  Full screenshot of ADView to explore four clades .............. 202
Figure A.7  Support for the most popular hylothesis (LP + ZYGN) ......... 203
Figure A.8  Pairwise comparison with the conflicting trees ....................... 204
Figure A.9  Combination of user-specified clades ................................. 205
Figure A.10  Exploration of low-support branches .................................. 206
Figure A.11 Screen shot of ADView exploring early lineage of LAND PLANTS208
Figure A.12 Previous hypotheses about early diversification of LAND PLANTS208
Figure A.14  Screenshot of ADView for exploring PROTOSTOMIA and DEUTEROSTOMIA ................................................................. 212
Figure A.15  Exploration of some discordant trees ............................... 213
Figure A.16  Pairwise comparison to locate potential causes for the discordance ................................................................. 214
Figure A.17  Screenshot of ADView for exploring ECDYSOZOA and LOPHOTROZOA ................................................................. 216
Figure A.18  Cluster ADs showing distributions of ECDYSOZOA ............. 217
| Figure C.8 | Performance threshold curve for threshold \( t \) | 254 |
| Figure C.9 | Labeling interface on a scene creation session | 256 |
| Figure C.10 | Analysis of segmentation results for session S12 (upper part) | 258 |
| Figure C.11 | Analysis of segmentation results for session S12 (lower part) | 259 |
| Figure C.12 | Analysis of segmentation results for session S13 (upper part) | 261 |
| Figure C.13 | Analysis of segmentation results for session S13 (lower part) | 262 |
| Figure C.14 | Analysis of segmentation results for session S14 (upper part) | 264 |
| Figure C.15 | Analysis of segmentation results for session S14 (lower part) | 265 |
| Figure C.16 | Full screenshot of a rule-based model on S1 | 267 |
| Figure C.17 | Partial screenshot of a rule-based model on S3 | 267 |
| Figure D.1 | Abstracted versions of concrete tasks for two datasets | 269 |
| Figure D.2 | Choices of node colors | 270 |
| Figure D.3 | Highlight nodes with specific properties | 270 |
| Figure D.4 | Search a node by its label | 271 |
| Figure D.5 | Latent space with bundled edges | 271 |
| Figure D.6 | Corgi paw-like menu for selected nodes | 272 |
| Figure D.7 | UpSet style alternative design for K-hop topology | 273 |
| Figure D.8 | Partial adjacency matrix | 274 |
| Figure D.9 | Count overview of neighbor set intersections of focal nodes | 274 |
| Figure D.10 | Full screenshot of CorGIE on the Movie dataset | 276 |
| Figure D.11 | Check overview of the Movie scenario | 277 |
| Figure D.12 | Check movie recommendations | 278 |
| Figure D.13 | Explore overview of Cora dataset | 281 |
| Figure D.14 | Find problematic node pairs in the Cora dataset | 282 |
| Figure D.15 | P1 explored the decision boundary in the Cora dataset | 283 |
| Figure D.16 | The USAir dataset without focal nodes | 285 |
| Figure D.17 | Find buggy node pairs | 286 |
| Figure D.18 | Screenshot of P2 focusing on a buggy node pair | 287 |
Acknowledgments

First of all, I would like to express my deepest appreciation to my supervisor Tamara Munzner. I met her for the first time in a remote interview when I was applying for UBC back in February 2015, and I was impressed by her kindness and wisdom. It has been six years since I first boarded the flight to Vancouver, and I am still impressed. I am impressed by her great interest in novel ideas, her attention to detail throughout a research project, and her hard-working towards a high academic standard, which has greatly influenced my taste of research. She taught me how to conduct scientific research with rigor, how to reason about visualization systematically, and how to think and write critically. She is a coauthor of all four papers described in this dissertation, and I enjoy working with her a lot. She is kind and caring to me in daily life, especially for allowing me to work remotely (even before the pandemic) to maintain my long-distance relationship between China and Canada. It is hard to express my gratitude completely with mere words, but there is a Chinese phrase, “生命力的贵人” (my best translation: life-changing savior). It is said that there could be only three in one’s life. I believe Tamara is one of the three for me.

Many have helped me in my academic journey.

I thank my PhD committee, Giuseppe Carenini and Michiel van de Panne, for their support and guidance on the trajectory of my PhD program. I thank the examiners of my dissertation, William Evans, Sathish Gopalakrishnan, and Enrico Bertini, for their thoughtful comments, which helped improve this document. I thank Xiaoru
Yuan, my advisor in my undergraduate study in Peking University, for introducing me to the field of visualization and helping me in job-hunting at the end of my PhD program.

I have enjoyed the collaboration with my paper coauthors: Tamara Munzner, Shing Hei Zhan, Takayuki Itoh, Jessica Dawson, Zhicheng (Leo) Liu, Jürgen Bernard, and Yang Wang. Shing, as the biology expert in ADVIEW, taught me biology and spent plenty of time with me to solve the phylogenetic problem. Takayuki is a visiting professor of our lab in 2018, who has a common research interest with me. We worked on the graph layout problem in SPRAWLTER. Even though I have never met Jessica in person, her work on the graph readability directly motivated and guided the SPRAWLTER project. Leo was my mentor when I was an intern in Adobe Research in the summer of 2018. He guided the LOGSEG project and provided lots of resources. Jürgen and Yang provided lots of input when we worked together on the CORGIS project.

I have enjoyed interacting with the members of the UBC InfoVis reading group: Dylan Dong, Matthew Brehmer, Anamaria Crisan, Kimberly Dextras-Romagnino, Madison Elliott, Georges Hattab, Shannah Fisher, Steve Kasica, Francis Nguyen, Jürgen Bernard, Mara Solen, and Ben Shneiderman. I enjoyed hanging out with them. I appreciate their feedback for my research and especially my manuscripts.

I thank many others who have provided feedback or inspirations for my research, including Sarah Otto, Sean Graham, Wayne Maddison, Daniel Archambault, David Auber, Fabian Beck, Hamish Carr, Donghao Ren, Bowen Yu, Meng Qu, and Muhan Zhang. I thank all anonymous reviewers of my papers.

I am grateful for the financial support from Natural Sciences and Engineering Research Council of Canada (NSERC), Adobe, Uber, and UBC.

I thank the administrative staff of the Computer Science department, especially Joyce Poon and Laura Selander, for helping me with the paper work during my PhD program.

I thank my families and friends for supporting me in various ways. My cousins
and their families who live in Vancouver have kindly invited me to holiday dinners and offered help since 2015. Michael Chen and Hanchen Hou were my lovely housemates for more than four years. Michael was my badminton and snowboarding buddy, and my guitar teacher. Chenglong Wang played video games with me while gossiping about academics and discussing interesting research ideas. My landlord Jack Song and his kind family provided me with cozy and affordable housing. They kept me sane in my busy PhD life.

I thank my families for their unconditional love and support over the years: my father Peixin Liu, my mother Xueping Lv, and my elder brother Zijian Liu. They supported my decision to pursue a PhD degree even when we were under enormous financial stress. I thank my uncle Huanlun Liu for his financial and emotional support.

Lastly, I thank my girlfriend and best friend, Luyao Zhang, for her love, care, and company in this long journey despite the fact that we were in different countries and thousands of miles apart physically.
Chapter 1

Introduction

In the digital era today, data, or, digitalized information, is collected with a growing scale in many domains, which also grows the need to analyze data. Data analysis is the name for this ubiquitous process to retrieve useful insights and inform decision-making. It is a broad research topic with many aspects and approaches. In this dissertation, I would like to distinguish between two kinds of approaches for data analysis: automation and human-in-the-loop. Automation, also known as end-to-end solution, is usually convenient and fast in reaching the analytical goal, but it only works in certain conditions. The data needs to be cleaned or transformed to a dedicated format, the analytical task needs to be defined clearly, usually in the mathematical form, and the stakeholders needs to trust the data, automation process, and the output. For example, to obtain the correlation of two scalar variables in a clean dataset, a statistician can click a button in software to compute their Pearson correlation value. However, most real-world problems do not come with clean data, and they are not always clearly defined. Even in the example of computing correlation of two variables, a statistician can understand and trust Pearson’s correlation, but a random person on the streets might not understand it, or a professor might not trust the student’s calculation in a quiz. Cases with raw data, fuzzy tasks, or low trust usually require stakeholders to intervene in this process manually, or, require human-in-the-loop analysis.
Figure 1.1: Munzner’s nested model [124] showing four layers of visualization concerns.

Visualization makes graphical representations from data for users and allows them to interact with the graphics. It has become an important tool for human-in-the-loop data analysis. Visualization accelerates the analytical process by reducing human cognitive load, helps to obtain a deeper understanding of data by revealing hidden patterns from aggregation, and communicates better to stakeholders with intuitive graphics.

1.1 Four layers of visualization concerns

Munzner proposed the nested model that contains four layers of visualization concerns [124], as shown in Figure 1.1, which I will borrow to describe how to systematically reason about visualization for human-in-the-loop data analysis.

In the nested model, we, as visualization researchers, first characterize the domain of the targeted real-world problem. We then abstract the data that need to be analyzed and tasks of stakeholders (users) as design requirements. Based on the design requirements, we design a suitable visual encoding to map the data to graphical representations, and interaction to link multiple representations. We finally devise effective algorithms to implement the design. The model describes a cascading relation between layers: the output of an upstream layer is used by the downstream layer, and a downstream layer supports the upstream one. Therefore, it often re-
quires solutions for the current and downstream layers to address the challenges in an upstream layer.

1.2 Challenges

I will situate the challenges that we face in the reported projects in this model. In general, the outermost layer is the closest to users, while the innermost one is the farthest. The motivation of our work is sometimes ignited by domain experts, and sometimes by the need of a visual encoding. We then identify the challenges in this igniting layer as well as the downstream layers.

In project ADVIEW, we target the domain of phylogenetics in biology. The discipline of phylogenetics studies the relationships between species and genes, which are often represented with tree data structures. For the data and task layer, biologists are capable of obtaining hundreds or even thousands of trees, and they need to compare these many trees in order to retrieve biological insights. The tasks of interest involve multiple granularities in a dataset, ranging from the whole tree collection, to a leaf node. For the encoding design layer, previous visual representations are either not efficient enough in terms of pixel real estate to handle the scale of data, or do not address the granularities required by the tasks. For the algorithm layer, we need efficient algorithms to deal with the scale of data, that is not only about the number of trees, but also the number of subtrees and leaf nodes.

In project SPRAWLTER, we identify challenges in the visual encoding layer related to node-link graphs. A node link graph represents nodes as points and links as connected lines, and it is the most popular visual representation for graph data. The positions of nodes are determined by a graph layout. A good graph layout must be readable by users, thus making readability an important metric for graph layout. Previous readability metrics do not account for different levels of visual saliency in graph data. For example, salient metanodes in multi-level graphs and variable node sizes in single-level graphs are ignored in previous metrics; small nodes that

\[^{1}\text{I will use the pronoun we to reflect collaborative nature of the work in this dissertation unless specified otherwise.}\]
are hardly visible (no visual saliency) are not penalized. For the algorithm layer, the remedy for these problems should be able to handle large graphs without too much computational overhead compared to previous ones.

In project LOGSEG, we work in the domain of image editing software. Typical image editing software like Photoshop is powerful but complicated. For the data and task layer, developers seek to understand how the users use their product, typically by collecting and analyzing user action logs. These logs could be huge, as a skilled user is able to perform hundreds of actions in a short period of time. Many user tasks require an underlying segmentation of the log sequences, that is, dividing the logs into meaningful chunks of user actions. However, the challenges are not only about the large scale of data, but also the variety of user tasks that rely on segmentation: different tasks require different segmentation granularities. For example, smart undo, a handy function allowing users to dial a group of actions back at once, requires a low-level chunking, while managing an overview of work milestones requires a high-level chunking. For the algorithm layer, the segmentation method should run fast enough as a backend thread without interfering with the regular functionalities of the software.

In project CORGIE, we identify challenges in the domain of graph machine learning. Graph neural networks (GNNs) are a class of powerful machine learning tools that model node relations for making predictions of nodes or links. For the data and task layer, GNNs leverage the topology and node features in an input graph to produce a node embedding, a common middleware byproduct, and then make predictions using the embedding. To evaluate the model, developers focus heavily on the final predictions, but they ignore the common data middleware of a GNN, the node embedding. The missing opportunity to connect the node embedding to its input graph makes it difficult to understand whether the GNN has truly learned from the input graph as expected, for example, whether different levels of node structures are preserved in the node embedding. For the encoding design layer, previous graph visual encoding cannot reveal the neighborhood structures for specific nodes.

To avoid confusion in this dissertation, we use “task” for analytical user task, and “machine learning application” or “downstream application” for the prediction.
of user interest, but information about topological neighbors of nodes is crucial in what GNN learns. For the algorithm layer, similarly to the other projects, we need to support graphs with many nodes and edges and ensure a usable interactive speed for users.

1.3 Multi-level structures

From the four different problems described above, I identify a common challenge that relates to data in multiple levels of detail, or, multi-level structures. As illustrated in Figure 1.2, for a dataset consists of many data items, we construct nesting subsets of data items according to the user tasks, resulting in a hierarchy, or, multi-level structure.

For hierarchical data such as trees and multi-level graphs, we can exploit the levels that come with it. In the biology project ADVIEW, the relevant levels from top to bottom are the whole tree collection, subsets of trees, an individual tree, a subtree, and a branch or a leaf node. In SPRAWLTER, we aim to take into account the multiple levels of visual saliency in readability metrics of node-link graphs. Multi-level graphs come with a node hierarchy, which we can directly exploit.

For non-hierarchical data, we can compute the levels to serve the user tasks. In the software usability project LOGSEG, we need to segment the user action logs in multiple levels for different user tasks. In the graph machine learning project CORGIE, we identify a missing opportunity in the evaluation of GNNs to help users understand the structural preservation of nodes in different levels of detail from the
whole graph to a node neighborhood.

Through these projects, I find that multi-level structures are ubiquitous in many domains. In complex real-world problems, the analytical tasks often require users to explore the multi-level structures in data, which could be either hierarchical or non-hierarchical.

1.4 Contributions

In this dissertation, I contribute visualization tools to the target domains in the four projects, as well as visual techniques and algorithms to the visualization community. These contributions exemplify how to visualize the multi-level structures in data for human-in-the-loop data analysis. I summarize the contributions of each project as follows.

**ADView (Chapter 2)**

- The data and task abstraction for comparing phylogenetic trees.
- A new visual encoding, the Aggregated Dendrogram (AD), to compress a dendrogram into a small resolution-aware visual representation showing topological relationships between subtrees of interest.
- An interactive visual tool, ADView, to compare one phylogenetic tree against a collection of hundreds of other trees using multiple linked views.
- An evaluation with multiple methods validating the utility of AD and ADView.

**Sprawlter (Chapter 3)**

- An area-aware approach for three families of clutter metrics: node-node, node-edge, and edge-edge crossings.
The sprawlter metric to capture the trade-off between sprawl and clutter.

An evaluation of the two metrics by comparing to traditional count-based metrics on many graph layouts.

**LogSeg (Chapter 4)**

- A multi-level segmentation model for image editing tools, where the low-level segmentation works reasonably well for smart undo, and the adjustable high-level one is based on the low-level one.

- An analysis of feature relevance to log segmentation, where we find that command, timestamp, and layer are relevant attributes, but image content is not as useful. We are the first to use layers for segmentation and confirm their relevance.

- An evaluation of the low-level chunks showing our model generalizes across three image editing scenarios, and a preliminary analysis of the suitability of higher-level chunks for these three scenarios.

**CorGIE (Chapter 5)**

- The data and task abstraction for visually exploring the correspondences between an input graph and the middleware node embedding, to understand if a GNN model has learned important characteristics of the input graph to construct the embedding.

- An interactive multi-view interface, CorGIE (Correspondences between a Graph and Its Embedding).

- A new graph visual encoding, the K-hop layout, to reveal how a GNN aggregates information from node neighborhoods.

In Chapter 6 I reflect on the multi-level structures in these projects, and the collaboration with many coauthors, before concluding this dissertation.
1.5 Evaluation methodology

I summarize the evaluation methods used in these projects by layers in Munzner’s nested model, as solutions for different layers require different methods. Figure 1.3 shows a summary of the evaluation methods I used in the four projects.

To evaluate the data and task abstraction, we interview domain experts on whether the characteristics of data and the identified tasks actually match their needs and fit their mental model. We review literature that describes the data and tasks in both the target domain and the visualization field. In SPRWLTER and LOGSEG, we compute quantitative metrics and inspect results qualitatively. In ADVIEW and CORGIE, after implementing the visualization tools, we demonstrate them to the domain experts (users) and get direct feedback, we test their utility in demonstrative usage scenarios ourselves, and we also recruit domain experts to use them on their own datasets. As these methods require the implemented tools, they also evaluate the downstream design and algorithm layer. Note that in SPRWLTER, the users of node-link graphs could be anyone with basic visualization training, and thus we do not need external domain experts.
To evaluate the visual encoding and interaction design, we discuss the design rationale, that is, to reason whether the visual marks, channels, and layouts can address the data and tasks in the context of the visualization literature. We compare the proposed design against alternatives in terms of the suitability and scalability of the data and tasks. The visual design should be suitable for the kinds of data (e.g., scalar vs. categorical values) and for the tasks (e.g., showing aggregated summaries of interesting data points vs. their identities). The visual design should be scalable to the number of data points and the number of data subsets in the levels of detail required by the tasks.

To evaluate the algorithm implementation, we focus on time and memory usage. We perform complexity analysis with the canonical big O notation, and run the algorithms to measure elapsed timings and memory consumed on my computer. ADVIEW and CORGIE, as web-based tools, are measured on browsers, while SPRAWLTER and LOGSEG are measured with Python scripts.
Chapter 2

ADView: Aggregated Dendrograms for Visual Comparison Between Many Phylogenetic Trees

We address the visual comparison of multiple phylogenetic trees in evolutionary biology. Specifically, we compare one reference tree to a collection of dozens to hundreds of other trees. We abstract the domain questions of phylogenetic tree comparison as tasks to look for supporting or conflicting evidence for hypotheses that requires inspection of both topological structure and attribute values at different levels of detail in the tree collection. We introduce the new visual encoding idiom of aggregated dendrograms to concisely summarize the topological relationships between interactively chosen focal subtrees according to biologically meaningful criteria, and provide a layout algorithm that automatically adapts to the available screen space. We design and implement the ADView system, which rep-

resents trees at multiple levels of detail across multiple views: the entire collection, a subset of trees, an individual tree, specific subtrees of interest, and the individual branch level. We benchmark the algorithms developed for ADView, compare its information density to previous work, and demonstrate its utility for quickly gathering evidence about biological hypotheses through usage scenarios with data from recently published phylogenetic analysis and case studies of expert use with real-world data, drawn from a summative interview study.

2.1 Motivation

Biologists who study phylogeny, which aims to resolve evolutionary relationships among all living and extinct organisms, often find it difficult to make sense of large collections of phylogenetic trees. These evolutionary trees are typically computationally inferred from living organisms. The ability to gather genetic data quickly and cheaply has led researchers to generate and analyze more and larger trees [101], including intermediate trees that arise within a computational pipeline for reconstructing a final tree. They frequently need to inspect these trees to consider their plausibility and meaning with respect to existing domain knowledge. They do not blindly trust automatic computational metrics, especially when dealing with unknown genes and species or when developing new concepts and methods. Current visual analysis tools fall short of their need to make comparisons between large number of trees at multiple levels of detail.

In the visualization community, visual comparison of trees has been studied for years [61], but existing visualization techniques do not scale to the current needs of phylogenetic researchers. We characterize scalability of visual tree comparison in terms of three factors: the number of nodes per tree, the number of trees being compared, and level of detail at which the internal structure needs to be shown. Different tasks require very different levels of detail. These factors have separable requirements with respect to screen real estate and human cognitive ability: techniques that are suitable for scaling up in one of these may not support the others. For example, a technique for inspecting clusters made of hundreds of trees does not
typically support fine-grained inspection at the subtree level within each one. Most scalable visual tree comparison work has been in support of comparison between a very small number of large trees. Approaches such as TreeJuxtaposer [126] are able to show highly detailed local structure, such as all descendants of a specific subtree. A few systems tackle the inverse problem of handling a large number of trees, where each tree is shown as a single point [77]. In this case, where a global distance measure is used to lay out those points, the available level of detail is an overview of the entire collection of trees. Between the two extremes, there is some work comparing dozens of trees at varying levels of detail, in many domains ranging from phylogenetic trees [19] to natural language processing [191].

Modern phylogenetic research takes place at a variety of scales: the number of trees can range from one to over ten thousand trees, and the number of nodes per tree ranges from a dozen to millions. The visual comparison of phylogenetic trees also encompasses a broad and complex spectrum of tasks. The research questions can range from assessing the overall uniformity or variance across a large collection of trees, to the exact placement of a particular leaf node across a small set of trees. Questions may pertain only to topological relationships between or within subtrees, or only to the distribution of attributes associated with nodes or branches, or to the combination of both. The problem space is huge; we worked with a group of phylogenetic researchers to understand their specific unmet needs.

We present ADView, an interactive system to visually compare one tree against hundreds of other trees, showing multiple levels of detail across multiple linked views. We propose an innovative technique to compress a tree into a concise representation that we call an **aggregated dendrogram** (AD), which captures the major topological relationships between several subtrees that are the focus of user interest. Our best-effort layout algorithm yields an AD that can adapt its size and the amount of presented information according to the available screen space. We cluster similar ADs to generate a visual summary of topological relationships between chosen subtrees that reveals information across multiple scales for datasets of hundreds of trees, enabling more detailed comparison than previous tools can achieve. ADView also includes a view showing distributions of trees in terms of
their agreement and conflicts of leaf memberships for the chosen subtrees. It has an on-demand view for pairwise comparison in full detail, and auxiliary views for attribute values and other information. We show the results of using ADView on real-world datasets, including usage scenarios based on published datasets and case studies arising from an interview study with five domain experts.

In summary, the contributions of this chapter are:

- task and data abstractions for the comparison of phylogenetic trees;
- an interactive visual tool to compare one phylogenetic tree against a collection of hundreds of other trees using multiple linked views;
- a technique to compress a dendrogram into a small resolution-aware visual representation showing topological relationships between subtrees of interest;
- validation through algorithm benchmarks, usage scenarios, formative expert feedback, and a summative expert interview study that provided case studies.

2.2 Related work

We discuss the visual representation of trees in general, in the biology domain, the many approaches to tree comparison, and task abstractions for trees and networks.

2.2.1 General tree visualization

Tree comparison falls within the very active research area of general tree visualization. The treevis survey by Schulz [155] enumerates hundreds of general techniques to visualize single trees, but they do not suffice for our chosen problem of tree comparison, which is more complex than displaying one tree[57]. The space efficiency of many tree representations is studied by McGuffin and Robert [118].
We were inspired by their efforts when designing the aggregated dendrogram. Previous approaches to the compression and simplification of general trees and graphs according to their statistical properties [9, 120] do not suffice to support our targeted tasks for phylogenetic analysis such as preserving monophyletic groups. The DOITrees approach [71] dramatically simplifies trees and like ours is a focus+context technique that uses elision rather than distortion, but does not support these targeted phylogenetic analysis tasks. Pretorius et al. [138] propose a visual encoding for cell lineage visualization, but it is focused on different requirements such as emphasizing symmetry.

2.2.2 Biological tree visualization

Biologists routinely use domain-specific tools such as FigTree [146], iTOL [103] and Dendroscope [86] to visualize and annotate a tree during analysis, and to generate publication-ready figures, but these tools do not support comparison at all. Biologists who are capable of programming could use Python or R libraries such as DendroPy [165] and Ape [132]. These libraries are handy for flexible data manipulation such as trimming and computing an additional attribute, but fall far short of the kinds of visual comparison that ADView provides.

2.2.3 Visual tree comparison

We categorize previous literature related to visual tree comparison by three dimensions of scalability: the number of trees, the level of detail at which the data must be shown to accomplish the intended task, and the number of nodes per tree. Figure 2.1 illustrates the first two dimensions of this problem space. The vertical axis shows the number of trees partitioned into four categories: pairs, dozens, hundreds, and thousands. The horizontal axis shows the level of detail representing how many levels of detail a tool can show, in terms of three rough groups: a single point, simplified structure, and the full topology. We identify three major groupings: Few in Full, Dozens at Multi-Scale, and Many as Points.
The Few in Full category has been heavily studied. An extensive list of these systems appears the survey by Graham and Kennedy [61], and all of the entries in the InfoVis 2003 Contest on tree comparison fall into this category [136]. With only a small number of trees to handle, often just two, the most scalable previous systems can deal with a massive number of nodes per tree. For example, the TreeJuxtaposer system [126] supports detailed structural comparison of large trees up to several hundred thousand nodes. A similar system was recently re-implemented by Phylo.io as a web-based tool [151]. Some propose superposition to stack trees visually, such as the color coding of CandidTree [100] and the explicit encoding of Beck et al. [12]. Several systems proposed drawing lines connecting the same nodes on two trees [41, 79, 176]. Tree comparison is a special case of the more general problem of graph comparison. For example, Archambault proposes a pairwise comparison approach of hierarchically coarsening difference graphs [6]. These approaches by definition do not handle the larger collections of trees that we focus on in this work.

At the other end of the spectrum, the Many as Points systems handle hundreds or thousands of trees, with no limits on the number of nodes per tree. As our group name suggests, each tree is represented by a single point and thus these sys-
tems provide only a high-level overview of relationships between trees according to some kind of similarity metric. Hillis et al. employ dimensionality reduction to map trees to a 2D space [77], revealing only cluster structure. Strimmer et al. propose Likelihood Mapping to visualize likelihoods for different hypotheses within a triangular coordinate system [163], but this tool is tailored for testing hypothesized relationships between four sequences or species families, and does not address more general problems. Fangerau et al. [49] compute a signature value for each tree and arrange them along a Hilbert curve to show structurally similar trees, and Hess et al. [76] cluster trees into a hierarchy to make sense of different parameter settings used in tree generation. The general idea of reducing complex structure to points is also applied to dynamic graph visualization by van den Elzen et al. [171]. These approaches cannot show the multiple levels of detail required for our target tasks.

The Dozens at Multi-Scale category comprises systems that handle from 2 to roughly 100 moderately-sized trees of around 100 nodes per tree, where at least two levels of detail are provided. Graham et al. propose a multi-view system for showing trees in different levels of detail using treemaps, node-link diagrams and lists [60], but they handle fewer trees than ADView and only a subset of tasks in Section 2.4. In DAViewer [191], Zhao et al. propose a compact view to aggregate information along two directions: from root to leaves, and for nodes across the same tree depth. Although it addresses questions about distributions and errors, it is not suitable for comparing topological structures at the subtree level and their choice of dendrogram does not scale to collections of hundreds of trees. PhenoBlock [58] uses small multiples to visualize many patient phenotypes compared to a single reference tree. Their solution takes advantage of the fact that phenotype ontology is fixed across all trees, so it would not work for the dynamic situations we present in Section 2.4. TreeVersity2 [64] embeds bar charts in trees to visualize changes of attribute values on nodes, while Vehlow et al. [174] and Bach et al. [11] deal with temporal changes of trees over time. Li et al. [105] proposed a new tree compressed representation with the focus on node attributes. All these systems support different data abstraction than ADView.
Consensus trees [20] are traditionally used by biologists to reason about collections of trees, where the places of agreement are shown as a standard tree and the disagreements are indicated by collapsing multiple leaves together under a single parent. Although they can be effective to pinpoint sparse disagreement in only a few areas, when substantial differences exist within a large collection of trees this simple approach breaks down. The summary graph of multiple domain specific graphs [97] proposed by Koop et al. has a similar limitation. DensiTree2 [18] uses half-transparent fuzzy branches to imply uncertainty in regions of difference, but this visual encoding shares the limitations of conventional consensus trees: fine-grained comparison is impossible when there is too much variation.

The previous work from Bremm et al. [19] is the most similar to ADView. They compare multiple phylogenetic trees at both global and local scales via topological summaries, for dozens of trees with dozens of leaves per tree. They also propose a new distance measure to capture structural differences between subtrees. Their approach for reducing visual complexity of trees is to retain the full path from a selected node all the way up to the root and hide the others. In contrast, our aggregated dendrogram method for simplifying trees, which preserves focal subtrees and carefully chosen local context while hiding more distant upstream nodes, is more compact and scalable. Moreover, our simplification approach can automatically adapt to available screen space. We present a detailed comparison of our results to theirs in Section 2.8.2, showing their limitations in handling for our identified tasks. In addition, they require all species under study to be present in all trees; in contrast, ADView can handle missing species, a situation faced by our domain experts that commonly occurs in phylogenetic analysis.

2.2.4 Task abstractions for trees

Gleicher offers some structural thoughts on the general task of visualizing comparisons [57]. The three papers devoted to task abstractions for networks only cover a subset of the tasks that we present in Section 2.4. Lee et al. [99], Pretorius et al. [139] and Kerracher et al. [91] cover tasks such as lookup by name, finding at-
tribute ranges, and general questions about topology inspection. However, they do not address tasks such as finding corresponding branches or conflicting taxa across a tree collection, much less the questions very specific to phylogeny such as understanding monophyletic groups. Several previous papers that propose techniques for phylogenetic tree comparison do present abstractions for more specialized tasks, but none cover all seven of the tasks that we propose and none explicitly identify all five of the relevant levels of detail that we characterize in the data.

2.3 Phylogenetic tree data

We introduce some basic domain terms that pertain to phylogenetic tree data and present an abstract data specification.

2.3.1 Phylogenetic tree

A phylogenetic tree describes the evolutionary history of living organisms. Leaf nodes are called taxa, and are extant species that are being studied. Internal nodes represent common ancestors, usually inferred by statistical methods of phylogenetic reconstruction. Tree collections may arise from multiple reconstruction computations with different parameters or assumptions. A subtree is composed of all descendants beneath a branch; the set of all taxa underneath a specific branch forms a monophyletic group, also called a clade. There is a 1-to-1 correspondence between a branch, an interior node, a subtree, and a monophyletic group.

Tree collections also arise when inferring gene trees that represent the different evolutionary histories of individual genes; the synthesis of these multiple histories into a single overarching species tree requires extensive comparison between these trees.
2.3.2 Data abstraction

Phylogenetic researchers conduct comparison with respect to a single special tree that can be interpreted as their main hypothesis, where they are looking for evidence to support or invalidate the hypothesis from a collection of other trees. We denote the special hypothesized tree as the reference tree and the other trees as the tree collection. Both are required as input data for ADView. Two typical scenarios are either one species tree vs. its associated gene trees, or one species tree vs. multiple species trees generated by different methods or parameters. The leaf nodes in these trees come with names, but internal nodes are usually not labeled.

Each tree branch has three attributes, two original and one derived. The primary original attribute is a support value associated with the branch, which is generated as an (un)certainty measure of how confident the reconstruction program is in its decision that the underlying leaf nodes should be grouped together. Each branch also has the secondary original attribute of branch length, which measures a distance from its ancestors related to evolutionary time. Although none of our target tasks directly rely on branch length, these lengths can provide useful context for biologists and are also sometimes used for sanity checking the data.

While the entire group of these trees has substantial overlap in the set of taxa, in many datasets an individual tree will have missing taxa. For example, in a collection of trees about the evolution of animals, some trees might not have kangaroos while others might not have turtles. We assume that trees are all rooted and binary. We also assume that every leaf node is unique in a tree; in biological language, that there are no paralogs.

Many of the tasks presented in Section 2.4 require the derived data of corresponding branches: for a specific branch in the reference tree, we must find its counterpart of the most similar branch in each tree of the tree collection, as illustrated in Figure 2.2(b). For example, the monophyletic group exploration task (T5 in Section 2.4) hinges on whether there is an exact match with similarity exactly equal to 1.0 between a reference tree branch and a corresponding branch, or an inexact match where the similarity is strictly less than 1. This computation is similar to
the best corresponding node (BCN) proposed in TreeJuxtaposer [126]; algorithm details are provided in Section 2.7.1

We use corresponding branches to also derive a third attribute, the gene support frequency (GSF), on every reference tree branch: the percentage of trees in the entire tree collection with exact matches for that branch. This metric captures uncertainty information of a subtree in the reference tree with respect to a tree collection, where support values usually fall short; several other biological metrics, such as the internode certainty [159], have similar goals.

2.4 Task abstraction

The overarching goal in ADView is to compare a single reference tree to a collection of many other trees. We break down this goal into a set of seven specific tasks that were identified through multiple rounds of interviews with four biologists who work across multiple sub-fields of phylogenetic research including plant phylogeny, insect phylogeny, and mathematical modeling. We held weekly meetings with one of them, who specializes in plants and is a co-author of this chapter,
to allow frequent checks on the validity of our task abstractions and the utility of
our visual interface during incremental refinement. In addition, we reviewed the
biology literature on phylogenetic reconstruction and analysis.

In the language of Kerracher and Kennedy [90], we take a multi-strand approach
to task gathering, primarily via interviews with domain experts and secondarily
by deriving from literature. Two potential threats to validity did not apply in our
case: the domain experts were indeed available, we were able to maintain a focus
on task discussions, and skewing tasks towards a single domain is a feature not a
bug given our goal to target phylogenetics in particular. We addressed the potential
difficulties in introspection by conducting many rounds of elicitation and followup.

The seven tasks are:

- **T1:** Find subtree by **taxa names**. Biologists are typically familiar with the
  names of species in their study; an example is to find the subtree that consists
  of mammals.

- **T2:** Find branch / subtree with respect to an **attribute range** in the reference
tree. The two attributes of support value and gene support frequencies are
useful in suggesting interesting areas to investigate.

- **T3:** **Pairwise compare** one tree in the collection with the reference tree
  in detail. Allowing biologists to inspect a tree in full detail alongside with
  the reference tree allows them to make biological judgment calls about their
  findings. For example, they can see how species in a subtree are distributed
  in a target tree compared to the reference tree: whether there are any outlier
  species that are far away from others or scattered across the whole tree.

- **T4:** Inspect **topological relationships** between multiple subtrees. Figure 2.2(a)
  shows three examples of relationships: nested, separated, or sisters. A sub-
tree can be **nested** within another subtree. Otherwise, biologists would like
to know if they are **separated** or direct siblings, known as **sisters**. They also
want to know which subtrees are the closest to each other, and have a rough
sense of the topological distance between specific subtrees.
• T5: Explore the **monophyletic groups** (the complete leaf set of a subtree) in terms of agreement or disagreement between trees, as illustrated in Figure 2.2(b): do the taxa nested beneath a branch in one tree (the three leaves to the right of branch A) all fall beneath the corresponding branch in another tree? (They do for A1 in Tree 1 but not A2 in Tree 2.) How many and which trees agree with exact matches or disagree with inexact matches? Do they agree with each other? How many alternatives are there?

• T6: Investigate the **conflicting taxa** between disagreeing subtrees. Users want to know which taxa cause the conflicts, allowing them to notice connections with previous findings such as whether a particular family of species is known to be hard to resolve.

• T7: Assess the **corresponding branch attributes** to associate their values with the (dis)agreement for monophyletic groups. This task is lower priority, but the uncertainty measure on the corresponding branches can help biologists to gauge their confidence on their findings. For example, a corresponding branch from a conflicting tree with an extremely low support value might signal a spurious conflict.

The actual workflow of biologists is incremental and iterative, so these tasks can occur in any order. We provide a few concrete examples of tasks sequences drawn from their analysis process, and discuss them further in Section 2.8:

• T1 ↔ T2: sanity check on topology and attribute values of the reference trees, then identify interesting subtrees.

• T4 ↔ T3 ↔ T7: explore topological relationships, compare hypotheses, and support values distribution.

• T5 & T6 ↔ T3 ↔ T7: check agreement and conflicts of taxa memberships, inspect details of a specific tree, check support values.

These tasks led us to identify five relevant levels of detail: 1) tree collection, 2) a subset of trees, 3) individual tree, 4) subtree, and 5) individual branch or leaf node.
These five levels provide a new abstraction that is a more precise articulation of the problem space than the three levels illustrated in Figure 2.1 and informed the design of the Aggregated Dendrogram.

2.5 Aggregated Dendrogram (AD)

We propose a novel technique to aggregate tree representation that specifically addresses the difficult problem of topological relationships between multiple subtrees (T4) given hundreds of trees, and partially addresses the agreement of monophyletic groups task (T5). In this section, we indicate the motivation of the aggregated dendrogram, present its visual design, explain how similar ones are clustered, and share a glimpse of its evolution with a discussion of how earlier designs fell short.

2.5.1 Design requirements

In essence, the problem comes down to how to visually compress a tree with space requirements. First, the size of the compressed representation should be small to fit in as many trees as possible within a single screen. Second, the information contained in the compressed representation should adapt to the available screen space: given a larger area, it should convey more details than in a smaller one.

We determine what should and should not be compressed according to our task abstraction. Given a rooted tree and several focal subtrees, the critical information to show visually is the type of relationships between focal subtrees: either nested or separated and more specifically, (non-)sister relations (T4); also, whether the subtrees agree with the reference tree or not (T5). It can be helpful to also show distances between subtrees and more information about conflicting subtrees such as neighboring clades and placement of the conflicting taxa. Our design target is to support up to five focal subtrees, to provide a reasonable tradeoff between power and comprehensibility. Our investigation found that this number suffices for most phylogenetic questions. We design for a screen resolution of 1920x1080.
2.5.2 Visual design

Driven by a combination of the design requirements and inspiration from previous work [19, 71, 118], we decided to take a non-distorting focus+context approach to compressing a tree, where less important information is elided [125, Chap. 14]. Specifically, we aggregate the dendrogram representation of a tree, as the biologists are most familiar with this form. Figure 2.3 presents the final visual design of aggregated dendrogram (AD). In Section 2.5.4 we justify this design by discussing the shortfalls of several alternative designs that were confusing to the biologists; our task abstraction evolved iteratively, as the less successful attempts surfaced our misunderstandings by violating previously implicit assumptions of phylogenetic analysis.

A crucial design choice is to have each rectangular block in an AD always represent a subtree; that is, a monophyletic group in that tree. We also take care to visually distinguish whether each block is an exact match that agrees with a subtree in the reference tree, or an inexact match indicating disagreement, through different border styling.

The focal blocks represent the subtree beneath this tree’s branch that corresponds to a selected one from the reference tree. These blocks have thick black borders...
and labels and their height is linear to the number of taxa inside, subject to the hard legibility constraints discussed in Section 2.7.2. Context blocks are rendered as less salient small rectangles with a grey border.

All AD block colors are consistent with the reference tree, where the proportions of colors indicate the percentage of highlighted taxa in this subtree. Exactly matching blocks, which have solid borders, are always fully colored because they exclusively contain all taxa under the reference tree branch. Inexactly matching blocks, which have dashed borders, are sometimes partly colored due to other taxa that have intruded, as shown in the partial blue block in Figure 2.3. Sometimes color appears in non-focal blocks due to escaped taxa, namely matched taxa that are found outside the corresponding subtree, shown by the tiny blue and orange context blocks between blocks A and B in Figure 2.4(c).

There is only one kind of block that is not a subtree: the oval that can appear at the very bottom of the AD represents all missing taxa, as shown in Figure 2.3. Its rounded corners clearly distinguish it from the other rectangular blocks, and its border is dashed. The distribution of different highlight colors for taxa within the same block is shown with vertically layered rows.

Unimportant parts of the tree are collapsed, as indicated by the special slash marker on a branch. For example, in Figure 2.3, many evolutionary events along the path from the root to A’s direct ancestor are elided, as we see from the slash crossing the dotted branch. Users can specify a collapse threshold of topological distance from highlighted blocks to either their lowest common ancestor or their nesting block. For example, Figure 2.3 was made with the threshold set to 2; block C is more than 2 levels deeper than B, so everything between B and C is hidden. This threshold is a soft constraint in the legibility criteria discussed in Section 2.7.2.

Our approach depicts the topological backbone relationships faithfully. In Figure 2.3, the following information can be easily interpreted: A and B are parallel but not sisters (direct siblings); C is nested within B with distance greater than two; A, B and C are far from the root (distance > 2).
The AD algorithm, which we describe in detail in Section 2.7.2, adapts to available screen space automatically, as shown in Figure 2.4. The algorithm makes a best-effort attempt to fit in as much information as possible, in terms of labels and context blocks, while keeping visible the critical information such as selected subtrees and their topological relationships (as defined in Section 2.5.1).

The individual ADs can be sorted by global distance to the reference tree, or by local distance with respect to a selected subtree.

2.5.3 Visual summary: Cluster AD

Although the individual ADs are spatially compact, when the tree collection is sufficiently large there can still be more than will fit within a single screen. We cluster the ADs that share the same topological relationships (as defined in task T4) and just render one representative AD, as illustrated in Figure 2.5. By grouping ADs into clusters, we obtain a summary of all possible kinds of relationships among focal subtrees in the tree collection. The value of preserving individual views that are topologically equivalent is they might end up highlighted differently depending on what other selections are made, which would provide useful information (as discussed in Section 2.6.6).
We ignore all context blocks while performing clustering, so that differences considered unimportant according to our criteria do not result in scattered or even spurious clusters. We randomly choose one of the constituent ADs from a cluster AD as a proxy for the entire cluster. The normal solid coloring is changed to a gradient to indicate the different proportions of filled color across the blocks in all ADs of that cluster. The clustering algorithm is presented in Section 2.7.3, and the gradient coloring is documented in Supplemental Section A.2.

### 2.5.4 Design evolution

We tried many unsuccessful versions of aggregated dendrograms, four of which are shown in Figure 2.6: remainder, container, fine-grained, and frond; it also shows the final skeleton layout which was successful. Eliciting feedback about these versions led to more clarity about what underlying biological meaning should be preserved in terms of both the focal and the context parts of the visual encoding.

One design question that required extensive iteration is the semantics for visible
blocks in the diagram. Our first two attempts, the remainder and the container layouts, did not adhere to our current semantics that a rectangular block must represent a monophyletic group; that is, a complete subtree underneath a branch. We eventually realized that identifying monophyletic groups was an underlying subtask for the larger-scale biological questions that were being studied, so all alternatives where rectangles could represent non-monophyletic groups were confusing to the biologists. We then articulated T5 (monophyletic groups) as an explicit task during our iterative refinement of the task abstraction, which occurred in parallel with the design evolution.

The fine-grained layout did not scale past a few chosen subtrees. The frond layout also suffered from some scalability problems with excessive requirements for horizontal nesting, and moreover elided so much context that important questions were difficult to answer. The final skeleton layout was the result of several months of exploration in this very large design space of tradeoffs. Our final design features a careful mix of which parameters should be controlled by users, vs. hardwired, vs. adaptively computed.

2.6 ADView interface

We first give a brief tour of ADView, followed by the details about each of the views, and then discuss the view coordination in depth with respect to the five levels of detail.

2.6.1 Interface overview

Figure 2.7 is a screenshot of ADView comparing a reference species tree against a collection of 68 species trees generated with different phylogenetic reconstruction methods. In the Reference Dendrogram on the left, four subtrees have been selected by the user to be the current focal groups, labelled A, B, C, and D and highlighted in blue, orange, green, and red, respectively. In the Tree Distribution view near the top, each row shows how the tree collection can be grouped
Figure 2.6: Five versions of the aggregated dendrogram design in chronological order of invention. The first four were unsuccessful: remainder, container, fine-grained, and frond. The successful fifth layout, skeleton, built on lessons learned.

Figure 2.7: Screenshot of ADView comparing a reference tree (left pane) against a tree collection (right pane).
by agreement or disagreement with respect to each focal subtree. Below that, the *Aggregated Dendrogram* (AD) view consists of cluster aggregated dendrograms and individual ones, showing topological relationships between the selected focal clades. Currently the user has selected the second cluster AD, highlighting 22 trees with a brown background, and is hovering on the second segment of row B in the *Tree Distribution* view, highlighting 12 trees with a black border. There are three auxiliary views showing additional information for the trees.

### 2.6.2 Reference Tree and AD Views

The *Reference Tree* view in the left pane of ADView is always devoted to showing the reference tree at full detail. We visualize this important phylogenetic tree with a branching style dendrogram with name labels for all taxa. This format is very familiar to biologists, intuitively supporting the Taxa Name task (T1). The horizontal length of a branch either encodes the branch length attribute, or uniform lengths can be chosen if high variance in branch lengths would lead to visual artifacts. In the *Reference Branch Attribute* view at the top of that pane, we show the distribution of the two main branch attribute values, the support values from raw data and the derived gene support frequency (GSF). The pink range selection area controls which branches that match the chosen attribute range are highlighted in pink in the *Reference Dendrogram*, in support of the Attribute Range task (T2).

The crucial interaction for ADView is to select a subtree in the reference tree and inspect the support and conflicts in the tree collection shown in the *Aggregated Dendrogram* views, in support of the Topological Relationships (T4), Monophyletic Groups (T5), and Conflicting Taxa tasks (T6). When the user picks an interesting subtree / branch, that subtree is labelled and colored in the *Reference Dendrogram*, a new highlighted block representing it is added to all of the aggregated dendrograms, and a row is added in the *Tree Distribution* view. ADView also allows for selecting a user-specified taxa group to handle cases where the user is interested in an arbitrary set of taxa that is not a monophyletic group in the reference tree, combining multiple choices using a popup menu rather than selecting a
Figure 2.8: Selecting a segment in the Tree Distribution view shows dots for each taxon within its group in the reference tree, temporarily in place on hover (above) or right-aligned for comparison between persistent selections (below).

We provide a simple layer of automatic control for AD size that takes into account the number of trees that need to be drawn in each of the two views, the Individual AD and Cluster AD, to make intelligent use of the available screen real estate. The goal is to set a target size that preserves a reasonable amount of detail, ideally while fitting all of the ADs into a single pane without requiring vertical scrolling. When the number of trees is too large to fit even with small individual sizes, the leftover ones that do not fit within the pane are elided by default; the user can show them with the More button. The user can override this automatic size control by using a manual slider to indicate a specific desired height and width for all ADs. In both cases, the best-match AD layout algorithm is used to create layouts for the specified size. This size control mechanism was added late in the design process, after watching users miss opportunities to obtain better overviews.

2.6.3 Tree Distribution View

The Tree Distribution view at the top is focused on the Monophyletic Group (T5) and Conflicting Taxa (T6). Each row consists of multiple horizontal segments that represent groups of trees in the collection that agree with respect to the focal
subtree, as shown in Figure 2.8. A segment represents the subset of trees whose matching subtrees share the same taxa set with each other. The first segment shows the subset that agrees with the reference tree, which is marked with a circled R marker (R for reference). The following segments, sorted by frequency from left to right, are the conflicting subsets. Each subset indicates one hypothesis of the taxa set of a selected subtree, and thus the full row shows all alternatives of the taxa set that exist in the tree collection with respect to a specific branch.

When the user persistently selects or hovers over a segment, dot markers appear beside the taxa labels for that subtree in the reference tree dendrogram. In the top example of Figure 2.8, the user hovers over the middle segment of row B, and we see from the black dots that the taxa set for this subset of trees does not have SPIROGYRA SP. In the bottom example, the user has persisted the dot markers for the middle and right segments in row A, and is now hovering on the left segment. The persistent selection triggers the reference tree labels to move from ragged-right layout to be right-aligned, to allow easy comparison of multiple columns of dot markers. The middle segment (labelled 1) has two taxa absent compared to the reference tree; that is, their closest subtrees to A do not include the first two taxa. The rightmost segment (labelled 2), which is very narrow because it only contains a few trees, is missing nearly half of the taxa in A.

2.6.4 Pairwise comparison

Users can select a specific tree from the tree collection for Pairwise Comparison (T3) with the reference tree in a butterfly layout, as shown in Figure 2.9. This mirrored layout, where roots are on the outside and taxa labels are close to each other, is a familiar idiom for biologists. The highlighting is linked between the two trees, allowing users to locate taxa of interest in the target tree. This linkage provides a very salient way to quickly check whether a colored block selected from one side is contiguous (meaning that there is agreement between them), or dispersed on the other side (showing conflict for that selected subtree). The missing taxa are explicitly shown, and are integrated with the linked highlight color. This
Figure 2.9: Pairwise comparison between the reference tree (left) and a user-specified target tree from tree collection (right).
feature was added in a late iteration in the design process, after we realized the importance of providing awareness of specific missing taxa. The ability to inspect taxa membership from the tree distribution, as described in the previous section, is supported on both sides of the butterfly view.

Users can also select a subset of trees and create a consensus tree out of it (with extended majority rule [20]) to pairwise compare with the reference tree. This feature provides a convenient way to quickly get a sense of what the subset of trees roughly looks like without diving into each tree one by one.

2.6.5 Auxiliary views

There are three auxiliary views on the far right.

The Tree List view at the top shows a list of names of trees with linked highlighting for the tree selection. Sometimes trees are named according to conventions that reveal useful information such as the data source, method, parameters to generate the tree, and gene identification numbers.

The Tree Similarity view in the middle encodes relative distances between pairs of trees to indicate potential cluster structure, where each dot represents a tree. This view is inspired by many papers in the category Many as Points. Distances between dots correspond to distances between trees as computed by the t-SNE dimensionality reduction technique [172]. The default distance metric is Robinson-Foulds to show global relationships, or the user can choose a the inverse Jaccard measure that provides local relationships with respect to a chosen subtree. These measures are discussed further in Section 2.7.1.

The Corresponding Branch Attribute view is aimed at the task of that name (T7) and shows the distribution of attribute values of corresponding branches for a selected focal branch / subtree.
### 2.6.6 View coordination and visual encoding

Visual indication of the coordination between multiple views is a complex and constrained problem in ADView because of the many levels of detail we support. Supplemental Table A.1 provides a detailed breakdown of the most basic level of view coordination: what aspect of data is visually encoded (or not) across all levels of detail and all views. The size of visual elements varies enormously between the views: the region subtended by a single tree ranges from a small point in the Tree Similarity view to a medium-sized cell in the AD view; subtree blocks can be large in the Reference Tree view but small in the AD view.

This variability poses a difficult challenge for color coding because distinguishability depends on size. Nevertheless, given the power of color for showing categories and the desirability of color consistency across elements between views [145], we carefully designed a color palette in consideration of these tradeoffs. The five medium-saturation colors for the subtree highlight blocks are designed to be roughly equal luminance so that no focal subtree seems more important than the others, sufficiently salient for large regions in the Reference Dendrogram view and small regions in the AD view, against both the white default backgrounds and brown selected backgrounds. There is adequate luminance contrast for the black text label legibility, while maintaining distinguishability for the small highlighted pink branches that could occur within these blocks or against a white background.

We off-loaded some selection indicators to other visual channels, using additional point marks to show taxon selection in the Reference Tree, and border outlines showing tree selection for both points in the Tree Similarity view and cells in the AD views. Moreover, shape coding and line type coding is already used extensively in the AD design itself.

Figure 2.10 shows a closeup of the two legends that appear at the bottom. To summarize, linked highlighting is encoded in several ways: five medium-saturation block colors for subtrees (blue, orange, green, red, purple), both background color (brown) and border outlines (black) to indicate two forms of tree selection, line color for branches and region color within attribute range histograms (pink), and
2.7 Algorithms

We provide an overview of the ADView architecture, then present specific algorithms in detail.

We use a browser-server architecture for ADView. The user can submit a dataset using the Upload page in browser, as shown in Supplemental Figure A.26 and Figure A.27, which sends the raw tree files (in Newick format) to the server. A Python asynchronous pre-process worker in the server is then invoked as shown in Supplemental Figure A.28. It parses all trees, computes distances between all tree pairs and corresponding branches between each branch in the reference tree and each tree in the tree collection, stores them into a MongoDB database, and notifies the user upon completion. When a client session begins, the browser fetches data from a lightweight RESTful API, performs dimensionality reduction to the distance matrix using t-SNE, presents the data, and responds to user interaction such as updating the ADs when a new branch of interest is selected. We use React+Redux to manage state of data model, view, and user interaction in the browser, and we render almost all visual elements in SVG manually with some help from D3. The implementation is available as open source under an MIT license from...
2.7.1 Corresponding branches

Finding comparable counterparts between the trees, namely corresponding branches, is done in the server when the user uploads a new dataset. The corresponding branch is the most similar branch in each tree of the tree collection to a specific branch in the reference tree, analogous to the best corresponding node (BCN) proposed in TreeJuxtaposer [126]. An example is shown in Figure 2.2(b).

We define the similarity between a branch \( b \) in reference tree \( R \) and a branch \( b' \) in a tree \( T \) of tree collection as the Jaccard index between two leaf nodes sets beneath the two branches, excluding the two branches, excluding missing leaves:

\[
\frac{|b.taxon - T.missing| \cap |b'.taxon - R.missing|}{|b.taxon - T.missing| \cup |b'.taxon - R.missing|}
\]

where \( b.taxon \) and \( b'.taxon \) are the leaf node sets under \( b \) and \( b' \), \( R.missing \) and \( T.missing \) are the respective missing leaf node sets. Missing taxa are excluded for a less biased comparison, following general practice in phylogenetics research. Failure to exclude would lead to a record of both conflicting and missing signals, in contrast to our goal which is to focus only on the conflicting signals.

This measure, which captures set memberships, is an intuitive extension to the standard biological convention of Robinson-Foulds distance metrics for trees [150]. This set-based measure is simple to compute and is a good match with our task abstraction and our visual encoding and interaction design choices. We note that Bremm et al. [19] propose a more complex similarity metric that captures both topological structure and set membership, which they use to create histograms that convey aggregate information about mid-level structure; however, the details about what is different cannot be seen with this approach because they are aggregated away. Our approach communicates a great deal of detailed mid-level topological structure through the ADs, so we only rely on the similarity metric for a first stage.
of exploring whether trees agree on what taxa are in a clade.

We need to compute the similarity score of every branch pair between the reference
tree and tree collection, as shown in [algorithm 1]. We use a bottom-up approach for
the GetSimilarity() function. Supposing \( A = b . \text{taxa} - T . \text{missing} \) and \( B = b' . \text{taxa} - R . \text{missing} \), the algorithm bottleneck is to compute \( |A \cap B| \) when \( A \) is fixed. The leaf
set of an internal node is comprised of the leaf sets of its children. We can thus
calculate \( |A \cap B| \) by summing \( |A \cap B_1| \) and \( |A \cap B_2| \), where \( B_1 \) and \( B_2 \) are the leaf
sets of \( B \)'s left and right children excluding \( R . \text{missing} \).

**Algorithm 1:** Corresponding branches to every reference-tree branch

<table>
<thead>
<tr>
<th>Input</th>
<th>reference tree ( R ), tree collection ( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>all corresponding branches ( corr )</td>
</tr>
</tbody>
</table>

1. **for** \( b \in R \) **do**
2.   **for** \( T \in C \) **do**
3.     \( \text{targetSet} \leftarrow b . \text{taxa} - T . \text{missing} \)
4.     **for** \( b' \in T \) **in post order** **do**
5.         \( \text{similarity} \leftarrow \text{GetSimilarity}(b, R, b', T) \)
6.         **if** \( \text{similarity} \) **is better** **then**
7.             \( \text{corr}[b][T] \leftarrow b' \)

**Function** GetSimilarity\((b, R, b', T)\)

8. **if** \( b' \) **is a leaf** **then**
9.     // \( b'.\text{card} \): cardinality of \( |b'.\text{taxa} - R.\text{missing}| \)
10. \( b'.\text{card} = 0 \) **if** \( b'.\text{taxon} \in R.\text{missing} \) **else** 1
11. \( b'.\text{intersect} = 1 \) **if** \( b'.\text{taxon} \in \text{targetSet} \) **else** 0
12. **else**
13.     \( b'.\text{card} = b'.\text{left.card} + b'.\text{right.card} \)
14.     \( b'.\text{intersect} = b'.\text{left.intersect} + b'.\text{right.intersect} \)
15. \( \text{union} \leftarrow |\text{targetSet}| + b'.\text{card} - b'.\text{intersect} \)
16. **return** \( b'.\text{intersect}/\text{union} \)

**Algorithm analysis:** [algorithm 1] computes similarity of all branch pairs between
the reference tree and tree collection in \( O(N^2 \cdot |C|) \) time, where \( N \) is the number
of leaf nodes in a tree and \( |C| \) is the number of trees in the collection. Function
GetSimilarity() is amortized $O(1)$ as the checking statement in line 12 and 14 utilizes a hash table with random hash functions. The $O(1)$ similarity calculation is a substantial improvement compared to the $O(\log N)$ TreeJuxtaposer [126] approach.

2.7.2 Aggregated dendrogram layout

Every time a subtree of interest in the reference tree is selected or removed, we update the layout of all aggregated dendrograms, which is handled on-the-fly by the browser. Given a tree data object, a set of focal subtrees, and the AD parameters, we use a best-effort mechanism to generate an AD layout that adapts to the specified size. The user-specifiable parameters exposed in the interface are the height and width of the AD, the number of context levels, and whether to apply labels and colors. Several more algorithm parameters exist, and are illustrated in Supp-lemental Figure A.4. The block color is determined at a later step in our rendering pipeline; the layout algorithm described here only computes the position and sizes of blocks and branches.

Algorithm 2: Best-effort aggregated dendrogram layout

```
Input: tree, focalSubtrees, params
Output: AD layout or failure

1 AD ← generateLayout(tree, focalSubtrees, params)
2 while not isLegible(AD) do
3     params ← shrinkParams(params)
4     if not params then
5         return failureGlyph
6     else
7         AD ← generateLayout(tree, focalSubtrees, params)
8 return AD
```

As shown in Algorithm 2, we keep generating tentative layouts with different parameter settings until the latest one passes a legibility test or parameter choices are exhausted. The function generateLayout() compiles a list of blocks and then links them together with either normal or elided branches by a bottom-up traver-
The coordinates of blocks and branches are then calculated based on the block hierarchy and relevant AD parameters such as total width and height. The function \textit{isLegible()} takes a layout and determines whether the blocks and branches will be visible if drawn. The hard target is that focal subtrees need enough room to be colored; the soft target is to have enough room for labels as well, as a redundant encoding if space permits. A set of parameters will be “downgraded” from more sprawling to more compact by the function \textit{shrinkParams()} every time the legibility test fails until the lower limit of all parameters is reached. There are two cascading sets of flexible parameters that can be changed: 1) the number of context levels, a metric to control how many context blocks to show; 2) inter-block gaps, branch lengths, and block sizes. The function \textit{shrinkParams()} first shrinks the first set until lower limit is hit, then starts to shrink the second set. If legibility is not achieved, the algorithm returns an error, which causes the interface to show a failure marker to notify users that there is no hope of fitting an AD in such space, indicating that the size parameters should be turned up manually using the UI sliders to override the automatic size-control algorithm one layer above this layout algorithm.

One example of a failure case would be rendering 5 selected nested sub-trees in 30x30 pixels. The user could decide whether it would be more appropriate to select fewer subtrees (for example, if some uncleared selections were left over from previous analyses rather than being crucial for the current investigation), or simply to allocate more space to each AD and accept the tradeoff that all of the individual ADs do not fit within a single screen. While our automatic algorithms do quite a bit of work on behalf of the user, our strategy is a mixed initiative approach that relies on the judgment of the expert user.

\textbf{Algorithm analysis}: Once the subtrees are selected by a user, the layout of each aggregated dendrogram is computed independently. The number of trials per AD depends on both the characteristics of the input tree itself, the amount of context to show, and other size related parameters. Our empirical observation is that with the current settings of ADView this number is less than 10 and usually just 1 or 2. The computational time of this algorithm is hence that of the function
generateLayout(), which is linear to the number of nodes in a tree. We also limit the number of active colored subtrees of user selection to five, and therefore the layout only needs to compute the position of about (colored and non-colored) 20 blocks at maximum.

2.7.3 Cluster aggregated dendrogram

Algorithm 3: Cluster topologically identical ADs

<table>
<thead>
<tr>
<th>Input</th>
<th>AD_layouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Cluster_ADs</td>
</tr>
</tbody>
</table>

1. strings ← serialize(AD_layouts)
2. Cluster_ADs ← group(sort(strings))
3. Cluster_ADs ← sortByFrequency(Cluster_ADs)

After generating AD layouts for all trees, we cluster them into groups of shared topological relationships among focal subtrees, as shown in algorithm 3. The serialization of AD computes a string that looks very similar to the Newick file format [1], which maps topologically identical AD layouts to identical strings. The user can choose to further distinguish exact and inexact matches, and sister-group or separate relationships. For example, the AD in Figure 2.3 is serialized into a string “(A, (C)B)”. If exactness and sister-group relationship is considered, it becomes “(A−,(C+)B+)”, where + and − stand for exact and inexact matches, and / and | for non-sisters and sisters. We only consider focal subtrees and ignore everything else. After serialization, the clustering problem is transformed into a trivial one: grouping the same strings together.

Algorithm analysis: Because serialization only requires a traversal of AD blocks, and the number of blocks is usually very small (less than a dozen), the time complexity of algorithm 3 relies on sorting short strings. It is thus $O(n \log n)$, where $n$ is the number of ADs.
2.8 Results

We present empirical benchmarks for the computational performance of the algorithms, an informal comparison focusing on information density to the Bremm et al. [19] system, and a usage scenario illustrating several typical workflow steps for ADView. We briefly report on formative feedback from ten domain experts during the iterative design process, and then present a summative expert interview study on the utility of ADView including two case studies. The Supplemental Section A.5 and Section A.6 includes multiple full screenshots of the first usage scenario, a second usage scenario with over 1300 gene trees compared to a single species tree. There is also a video to show the look and feel of the system in action.

2.8.1 Computational performance

We deployed the server programs and database to a small server with two 2.20 GHz Intel Xeon E5-2650 CPUs and 2G RAM hosted by our university, and run the client on a Chrome browser on a 2012 Macbook Pro with a 2.50 GHz Intel i5 CPU and 8G RAM. For a dataset with 500 trees and about 100 leaf nodes per tree, created by sampling from the public 1KP dataset (as discussed in Section 2.8.3), it takes roughly 50 seconds to pre-process the data on initial upload and 10 seconds to load in browser at startup. The size of the compressed dataset is about 2MB. The user interaction in the browser is fluid when the number of trees is less than roughly 500. There is noticeable delay of about 1 to 2 seconds for datasets beyond this scale when all individual aggregated dendrograms are displayed, but immediate responsiveness can be restored by collapsing that pane. We used a caching technique to accelerate the rendering of AD in browser, which is detailed in Supplemental Section A.3.2.
2.8.2 Information density comparison

We directly compare ADView to its closest competitor, the tool developed by Bremm et al. [19], using their dataset of 33 trees and 32 taxa. An initial impression from ADView is that the trees seem very different from each other: very few branches in the reference tree are supported by majority. In the rest of this comparison we focus on the visual encoding and interaction scalability.

Supplemental Figure A.19 shows their system after adding 12 trees to the right panel and selecting one subtree of the reference tree, which is highlighted in pink. All matched nodes are highlighted across the trees, but their system only allows one focal subtree at a time. Without any manual filtering, many irrelevant parts are visible and the layout is visually crowded. Supplemental Fig.S21 shows the view after we filter out elements with similarity scores below 0.5. Substantial space is required for layout because the entire path between the internal nodes and the root is preserved.

Supplemental Figure A.20 shows ADView with the same dataset after we select three subtrees in the reference tree. (The paper’s materials do not include enough information to replicate the selected groups, so we randomly chose subtrees to highlight.) The cluster ADs show the major topological relationships between the blue, orange and green clades (task T4), and the Tree Distribution view shows the agreement and conflicts on each of the selected clades in terms of taxa memberships (T5). These two views in ADView provide topological information in a more space-efficient way than the full dendrograms in Figure A.19, and with far more detail than can be gleaned from their heatmap and histogram views based on similarity scores. If the similarity scores are of direct interest, in ADView the user can use sorting by similarity, see distributions in the histograms, or hover on a specific branch in the Pairwise Comparison view.

From the screenshots, we can conclude two major advantages of ADView over their system. First, one of the important tasks, topological relationships between subtrees (T4), cannot be supported by the color coding and filtering mechanism of the similarity score they propose, while the ADs in ADView are more suitable for
this task. However, a usability disadvantage of our system is that due to the broader scope of task complexity, it is more difficult to learn. Second, ADView can scale to much larger number of trees due to higher spatial efficiency of tree representation. We also note that they do not show taxa names on their interface, which is an essential piece of information for our biologists to interpret the display. However, for datasets such as this one with a small number of trees and small tree sizes, their system has the advantage of showing selectable taxa in all of the small-multiple tree views, which fit on a single screen side by side.

2.8.3 Usage scenario 1: 1KP pilot study

The 1000 Plants Consortium (1KP) gathered phylogenetic data for thousands of plants and published a pilot study about the origin and early diversification of land plants [183]. One of the main research questions is to determine the sister group of land plants (LP for short).

The authors picked 103 representative plant species of major groups in streptophytes, and produced 69 species trees by performing 69 different analyses; that is, 69 runs of the computation pipeline, spanning different statistical methods and parameter settings, different filtering criteria, and different kinds of sequences. They compared these trees with previous hypotheses to the research questions.

We took their published data and checked whether their research questions could be answered quickly with ADView. This usage scenario was created with the domain expert who is a co-author on this chapter, who has close ties to the researchers who wrote the original paper. We report on one of these questions below, and present a second one in Supplemental Section A.6. We randomly picked one tree as the reference tree, using the other 68 as the tree collection. We also confirmed that the same results were reproduced by picking another reference tree, where all but 1 of the 68 aggregated dendrograms remained very similar.

Correspondence of cluster ADs to known hypotheses: the previous hypotheses said land plants (LP) might be the sister of Zygynamatophyceae (Zygn),
Figure 2.11: Annotated screenshots for a research question in the 1KP pilot study: which is the sister group of LP? (a) Previous hypotheses (excerpt from their paper [183]) and the cluster ADs in ADView; arrows show hypotheses the cluster ADs support. (b) Consensus tree of cluster C3 pairwise compared to the reference. (c) Branch support values of trees supporting the LP+CHAR hypothesis.
CHARALES (CHAR) or COLEOCHAETALES (COL), as illustrated in Figure 2.11(a). We selected and highlighted these four clades in the reference tree as our focus. ADView generated the ADs and clustered them into five groups (without considering exactness but with sister-group relationships), which can be regarded as five hypotheses that this dataset supports. We quickly found out that clusters C1 and C4 support the ZYGN-sister hypothesis, clusters C2 and C5 support CHAR-sister, but cluster C3 does not agree with any of the three (task T4).

**Support values in C1 + C4 (ZYGN-sister trees):** when we checked the distribution of support values (T7) by selecting the trees of C1 and C4 and created a sub-collection, as shown in Supplemental Figure A.7, there are trees with low support values. We can interpret this display as a counter-evidence to the ZYGN-sister hypothesis.

**Conflicts in ZYGN:** inspecting the Tree Distribution view in Figure 2.7, we saw that all 68 trees plus the reference tree agree that LP, CHAR and COL are grouped together (monophyletic). However, a small portion of trees disagree with the reference tree about ZYGN, as shown in the hovered-over segment (T5). To investigate what triggered the disagreement (T6), we selected those trees by clicking on that segment, made a consensus tree out of them, and pairwise compared with the reference tree (T3), as shown in Figure 2.11(b). It turned out that a species, SPIROGYRA, was not included in ZYGN for these trees; we pairwise compared with each selected tree to double check (T3). The authors of the pilot study confirmed that there are complicated underlying issues for this misplacement.

**Details in C2 + C5 (CHAR-sister trees):** in Figure 2.11(c), we further dug into the support for CHAR-sister hypothesis by combining LP and CHAR into group Z to see whether trees contain it (T1). Since Z is not a complete subtree in the reference tree we could not directly select it, so we used the popup menu that supports selecting an arbitrary set of user-specified taxa to choose the combination of LP and CHAR. There were 23 exact-match trees, which is exactly the number of trees for cluster C2 plus that of cluster C5 (T4). In the Corresponding Branch Attribute view, we checked the black foreground bars in the histogram of support values for corresponding branches of these 23 trees (T7). The circled bars showed
that some trees have very low support for LP+CHAR (biologists consider support less than 0.5 very low). The Tree List view showed that these trees are mostly generated from a kind of unfiltered sequence data that has been called into question in the domain literature.

The workflow description above covers some interesting and representative steps extracted from the full session, in which further visual analysis was conducted to check the supporting and conflicting evidence for additional hypotheses. By the end of the full session, we had found substantial evidence to support the pilot study’s conclusion that ZYGN is probably the sister-group of LP, although there seems to be counter-evidence that merits further inspection.

2.8.4 Formative expert feedback

We obtained informal formative feedback during the iterative design phase from the four phylogenetic researchers who worked with us throughout the project. At a quite mature stage of development we also presented it to six other phylogenetic researchers via chauffeured demos. Overall we received positive feedback confirming that our task and data abstractions matched with their biological intuitions, and that the functionality provided by ADView would be helpful to them. We also gathered suggestions for improvement, many of which informed further iteration of the design.

2.8.5 Summative expert interview study

We conducted an expert interview study with five domain experts; our observations and their feedback served to confirm the validity of our task abstraction and interface design. We describe the participants and procedure, present two case studies here (with an additional one in Supplemental Section A.8), and discuss their feedback.
Participants

We recruited five participants who are actively involved in phylogenetic research and conducted four study sessions of 90 minutes each. In this chapter, we refer them as P1, P2, P3, P4a and P4b (the latter two were in the same shared session). Four of them come from different labs at our own university, and one from overseas. Four were researchers focusing on biological questions with different kinds of organisms, three of them PhD students and one principal investigator; one was an experienced bioinformatician whose daily job is to manage, process, analyze phylogenetic data of his lab. Two had seen a previous version of ADView demonstrated to them a few months before the study took place; one had seen screenshots accompanied by a general description. Participants P4a and P4b did not have relevant research datasets of their own and used those from the usage scenarios; the three others used their own data. Supplemental Table A.2 provides detailed participant characteristics.

Procedure

We requested they send us one or two of their own research datasets a few days before the session and added them to ADView ourselves to ensure they were ready for immediate use; in some cases we undertook minimal pre-processing effort such as mapping machine-friendly sequence labels into meaningful human-readable names. In the session, we first gave a 20-minute demonstration of the two usage scenario datasets to illustrate the system’s functionality, and also explained the exact details of how their individual datasets were pre-processed. Next, the participants used ADView on their prepared datasets for roughly 50 minutes, where they were asked to think aloud while working. Our primary goal with this study was to assess the utility of the system rather than to improve the software usability, so we encouraged them to immediately ask us any questions about the meaning of the visual encodings and how to perform certain operations, stepped in if we saw a clear misuse of the tool or implementation bugs, discussed possible follow-up actions if they were stuck, and asked questions if we were not able to understand what the partic-
Figure 2.12: Detail views of P1’s findings. (a) Reference tree with four focal subtrees selected to investigate the position of T10 (blue, label A) and a branch highlighted in pink (circled). (b) Support distribution histogram with very low values selected, driving the pink highlight in the reference view. (c) Cluster ADs showing the top-frequency clusters.

Participants were doing. In the final 20 minutes we asked a set of open-ended questions, documented in Supplemental Section A.4.2. We recorded audio and video for the entire session, and took notes manually.

Case study 1 findings

P1 compared a species tree against 260 gene trees of plants. We obtained permission to show this dataset with anonymized organism names to protect unpublished data.

Sanity check on support values: upon loading the dataset, she noticed branches with extremely low support values according to the reference tree attribute distribution, shown in Figure 2.12(b). She selected the leftmost bar, which highlighted one branch in thick pink in the reference tree (T2), shown in Figure 2.12(a). Once she hovered on that branch, which contains all in-group taxa, the branch detail panel
Figure 2.13: Cropped figures of P2’s findings: (a) The selected subtrees of interest (blue - Blastocystis Strain 1, orange - Blastocystis Hominis, green - Blastocystis Strain 5); (b) different positions of orange taxon across bootstrapped trees.

shows that both its support and length are zero. She clicked on this branch, and the Tree Distribution view showed that all trees agree with the reference tree, which seemed to conflict with the zero support value (T5). After checking a few gene trees in detail using the Pairwise Comparison view (T3), she figured out that the zero-support in-group branch was generated by the rooting process, and dismissed it as not relevant to her research questions.

Position of T10 She then highlighted branches with relatively low support values (0.8 to 0.9) in the same fashion (T2), and discovered a specific taxon T10 was placed differently across the gene trees. To understand the alternative positions of T10 within a bigger monophyletic group, she broke it down into several subtrees, visible as the blue, orange, green and red groups (T4) in Figure 2.12(a). The first 4 of the 56 cluster ADs are shown in Figure 2.12(c). The majority of the remaining clusters contained only one AD, which suggested the relationships between the focal subtrees were badly disputed. To investigate the reasons for the disagreement, she looked at the consensus trees of the clusters and pairwise compared several individual ADs to each other.
Case study 2 findings

P2 compared a consensus tree (used for the reference tree) against 100 bootstrapped trees of parasites, and approved our usage of this dataset with unsanitized labels.

Sanity check on known strains: to familiarize himself with the interface and interaction, he selected a few monophyletic groups of different parasite strains (T1). He was able to quickly confirm his knowledge of which two strains are closer than others, and the general branching pattern of strains 1 to 5, by selecting them in the reference tree and then checking the cluster ADs (T4).

Unknown outlier (orange): he identified an interesting species, BLASTOCYSTIS HOMINIS, shown as the orange group in Figure 2.13(b). He knew that it usually lives in pigs, but he saw that here it was placed within the group mostly related to humans (T4). He knew that the orange group branched with the green one because he had seen it in the reference consensus tree on the left, but he had not realized that there is uncertainty about placement of the orange group until he saw the cluster ADs. Figure 2.13(b) shows that only 40 trees out of 100 have the same branching pattern as the reference tree, whereas there are 29 trees suggesting that the orange is closer to the blue and that the green branched before the other two. He then constructed a sub-collection of each cluster and pairwise compared some trees one by one within each sub-collection against the consensus tree for more detail (T3).

Feedback

There was general agreement from all participants that many aspects of the design were successful. The task and data abstractions matched their biological intuitions, as did the results of the corresponding branches algorithm. The overall AD design showed many trees in a way that was considered quite intuitive, despite being a completely new visual representation to all participants. They all could see themselves using ADView in upcoming research projects to analyze the trees generated by a reconstruction pipeline.
Participants also noted limitations of ADView. Most thought it was complicated to learn, and they would need more experience using it to fully exploit its power. One minor point of confusion came at startup time when no subtrees are selected, where the ADs consist of only a single blank rectangle. While most participants agreed that the Tree Distribution view was useful for exploring agreement and conflicts of a selected subtree, it was sufficiently unusual that two participants did not fully understand its meaning until a discussion in the question period at the end. One participant observed its connection to an existing visualization that shows the distribution of alternative taxa groupings with a small pie chart next to a tree branch [159, Fig. 2]; she noted that we had “unrolled” a pie into a horizontal bar.

Although the main goal of this study was not usability testing, we did make design refinements after each session to address awkward interactions. Most were minor interaction improvements, but a notable late-stage addition was to automatically compute appropriate sizes for the ADs and pass that request to the layout algorithm, rather than rely on the user to explicitly set sizes through a manual slider.

2.9 Discussion

ADView can be used by researchers to speed up the overall process of phylogenetic analysis at multiple stages in a project, especially exploratory analysis right after obtaining the trees and also to generate figures to communicate findings near the end of projects. We envision a workflow where researchers would typically follow up their initial explorations with more rigorous statistical tools. It is difficult to document the amount of speedup quantitatively, since biology papers do not typically provide any explicit estimates of how long their original analysis process required, as with our usage scenario source [183]. However, our biologist co-author has extensive experience in analyzing this kind of data. His assessment is that the analyses in the source paper require multiple dedicated R scripts, which could take a few days or even weeks depending on the programming expertise of the biologist. With ADView, it takes only a few hours or less to conduct similar analyses. ADView provides substantial speedups in assessing the quality of data,
process debugging and other sanity checking, identifying interesting areas, gathering initial evidence, and generating hypotheses.

ADView can handle datasets of hundreds of trees with about 100 leaf nodes per tree from an algorithmic point of view. The de facto limit on the number of leaf nodes is primarily due to the design target of fitting the detailed reference tree dendrogram on a single screen without scrolling. There is no hard limit on the number of trees for our proposed method, but our current implementation on the browser can only deal with roughly 500 trees at interactive frame rates before the number of SVG elements would slow down the rendering process. However, for larger datasets, the user can filter out many trees and analyze smaller subsets of dozens to hundreds of trees, or simply close the Individual AD view and achieve fast response time by relying solely on the Cluster AD view. Addressing these algorithmic bottlenecks in future work would allow studies to determine the perceptual scalability limits of our approach.

2.10 Summary and future work

In this chapter, we provide task and data abstractions for the problem of one-to-many tree comparison in the domain of phylogeny. We propose a new technique, the aggregated dendrogram, to summarize the topological relationships between focal subtrees via a simple, intuitive, and effective representation. We provide an algorithm that adapts to available screen space. We present the design and implementation of ADView, which provides insight into a tree collection of hundreds of trees at five levels of detail: the full collection, a subset of trees, single tree, subtree, and individual tree branches. We validate our approach with empirical benchmarks, direct comparison to the most similar previous system [19], usage scenarios, and an interview study conducted with five domain experts including case studies with their findings from using the system.

Extending ADView to handle duplicate leaf nodes, which arise from common evolutionary events such as gene duplication, would be an excellent and challenging direction for future work. It would also be interesting to see what aspects of
ADView are useful in general tree comparison problems outside of the phylogeny domain.
Chapter 3

Sprawlter: Combining Sprawl and Area-aware Clutter for Graph Readability Metric

Graph drawing readability metrics are routinely used to assess and create node-link layouts of network data. Existing readability metrics fall short in three ways. The many count-based metrics such as edge-edge or node-edge crossings simply provide integer counts, missing the opportunity to quantify the amount of overlap between items, which may vary in size, at a more fine-grained level. Current metrics focus solely on single-level topological structure, ignoring the possibility of multi-level structure such as large and thus highly salient metanodes. Most current metrics focus on the measurement of clutter in the form of crossings and overlaps, and do not take into account the trade-off between the clutter and the information sparsity of the drawing, which we refer to as sprawl. We propose an area-aware approach to clutter metrics that tracks the extent of geometric overlaps between

node-node, node-edge, and edge-edge pairs in detail. It handles variable-size nodes and explicitly treats metanodes and leaf nodes uniformly. We call the combination of a sprawl metric and an area-aware clutter metric a sprawlter metric. We present an instantiation of the sprawlter metrics featuring a formal and thorough discussion of the crucial component, the penalty mapping function. We implement and validate our proposed metrics with extensive computational analysis of graph layouts, considering four layout algorithms and 56 layouts encompassing both real-world data and synthetic examples illustrating specific configurations of interest.

3.1 Motivation

Computing quality metrics is an important and popular quantitative approach to evaluate and generate visualizations [13], especially for node-link diagrams, an intuitive and effective visual representation for relationships between entities in a graph. Graph layout has been studied for many years, with major effort devoted to automatic generation of readable and faithful node-link graph layouts [56, 69]. One of the important aspects in making good layouts is to have good metrics to measure the readability or faithfulness, which are used by graph layout algorithms either explicitly or implicitly. In particular, we focus on readability metrics, also historically known as aesthetic criteria. Many readability metrics have been proposed in previous papers, such as edge-edge crossings, node-edge crossings, edge bends, and angular resolution. Many empirical studies have been conducted to evaluate their correspondence with human judgements [32, 143]. We note three problems with many previous readability metrics.

First, they report simple integer counts of detrimental events such as node-edge crossings. Although counts are straightforward to understand and to compute, they lack the precision to capture how badly an event hinders readability, such as how much an edge crosses a node as measured in length, or whether two edges cross with a glancing angle.

Second, they only compute on single-level structure, even in a multi-level graph layout, which are used in many domains [117], especially for large graphs. A multi-
level graph combines a cluster hierarchy with a base graph, where the original graph nodes comprise the lowest leaf level of the hierarchy, and metanodes (also known as clusters or subgraphs) at higher levels contain lower-level nodes. The metanodes are typically more salient than the leaf nodes because they are bigger in size, but this important multi-level structural information is not taken into account when measuring readability by the previous metrics.

Third, they are mainly clutter-only metrics that only penalize crossings and do not fully assess the layout quality. In addition to clutter, information density [125] (also known as data-ink ratio [170] or space efficiency [118]) also matters.

We use the term geometric sprawl, or sprawl for short, to capture disparity between the size of small features and a potentially large canvas on which they are spread out. This measure is different from topological sparseness: sprawl considers geometric layout information such as coordinate positions and area. There is an important trade-off between clutter and sprawl: it is possible to eliminate all overlaps (especially for node-node and node-edge overlap) by simply either pushing the nodes away from each other or shrinking the node size until they are completely separated, but at the cost of tiny nodes and edges spread out on a huge canvas. The quantification of sprawl itself is not new; it is the inverse (reciprocal) of metrics recently called compactness by Kieffer et al. [92] and visualization coverage by Dunne et al. [39]. The idea of minimizing the metric of total area is longstanding [36]. The novelty of our work lies in the direct capture of this readability trade-off, and our emphasis that it constitutes the most central property of graph legibility.

We illustrate the three problems above in Figure 3.1. All three example layouts show two-level synthetic graphs containing four colored metanodes drawn as minimum bounding circles of their constituent leaf nodes. The count does not reflect the fact that layout (b) is more cluttered than (a) in terms of both leaf node overlaps and metanode overlaps, as there are exactly six crossings in both layouts (the integer count problem). Nodes in layout (c) are shrunk and spread out on a larger canvas resulting in fewer overlaps, but the layout suffers from poor space efficiency [118] due to sprawl.
Figure 3.1: Comparison of approaches to measuring node-node crossing in three synthetic 2-level graph layouts. (a) Some overlap. (b) More overlap. (c) More sprawl. Our sprawlter metric \( T \) is the geometric mean of the sprawl \( S \) and area-aware clutter metrics \( A \). Clutter metrics \( A \) and \( C \) are broken down into metanode vs. leaf node pairs. For all metrics, larger numbers are worse, indicating less readability. The increased overlap between leaf nodes and the metanode pair is captured by the area-aware metric \( 24.03 > 15.91 \). The increased sprawl when nodes are shrunk in size and pushed away from each other to avoid overlap, resulting in a more sparse layout, is captured by the sprawl \( 37.50 > 8.45 \). Their mean \( (T) \) assesses the trade-off between clutter and sprawl in the geometric configurations.

To address the first two problems, we contribute an area-aware approach for three families of clutter metrics: node-node (NN), node-edge (NE), and edge-edge (EE) crossings. It measures the amount of geometric overlaps (NN, NE) or crossing angle (EE) in both single-level and multi-level graph layouts, then computes a penalty for each geometric measurement, and finally sums the individual penalties as a NN, NE, or EE clutter metric. We identify general requirements and make specific choices for the crucial step of mapping geometric measurements to penalties. To address the third problem, we contribute the sprawlter metric to capture the trade-off between sprawl and clutter. It computes the geometric mean of sprawl and area-aware clutter for each of the NN, NE, and EE families. We implement these new metrics, and validate their benefits by comparing quantitative computational results of our approaches to traditional count-based approaches and recent proposals [39], with respect to qualitative assessments of layout pictures.
3.2 Related work

We discuss related work on readability metrics and the evaluation of these metrics.

3.2.1 Graph drawing readability metrics

We identify two categories of graph drawing metrics: single-purpose and compound metrics. The former only quantify one single feature of the layout, while the latter explicitly combine or implicitly reflect multiple single-purpose ones.

Single-purpose metrics

The single-purpose metrics most relevant to our own work focus on how clearly the drawn elements of a graph layout can be seen. There are many other types of metrics that are less relevant, including those related to graph topology such as average degree of nodes or modularity, those related to faithfulness such as shape-based metrics [45], and those measuring specific aspects of perception and cognition such as symmetry [44, 140], edge bends [143], and edge continuity [182]. A recent paper uses many metrics to build a neural network to assess quality of graph layout, but does not focus on readability metrics [94].

Well over a dozen single-purpose readability metrics have been proposed, with some evaluated by controlled experiments with human subjects. Early work from Purchase formally documented several metrics [141], with many subsequent experiments that compare these and other metrics to human judgements [32, 44, 182]; Dunne and Shneiderman provide a survey of empirical studies on readability metrics [38]. We categorize these metrics by the three problems we note above: clutter, multi-level, and sprawl.

Clutter metrics. Clutter is an important and obvious readability obstacle. The well known clutter-related metrics are edge-edge, node-edge, and node-node crossings, which report discrete integer numbers of the crossing events. Some metrics re-
port continuous real numbers, such as angular resolution at nodes [141], which measures the minimum angle of edges that are incident to an individual node, and angular resolution at edges, which measures the angles between crossing edges, or the total resolution of Argyriou et al. [8] that combines them.

The impact of crossing angles, namely the angle at which two edges cross, has been heavily studied. Although Ware et al. did not find significant relevance between human performance and average crossing angles in their multi-factor experiment using global graph layouts as stimuli [182], a later single-factor experiment using simple synthetic drawings conducted by Huang et al. showed that the crossing angles have a significant impact on response time [83]. Dunne et al. proposed another version of edge crossing angle metric, defined as the average deviation of the crossing angles from an ideal angle (70°) [39]. Their metric is 0 for the ideal crossing angle, which ignores the fact that there is still a detrimental crossing event that reduces readability (R2 in Section 3.4). Our own edge-edge metric differs with an explicit function that maps angles into penalties instead of using them directly, allowing us to clearly distinguish even ideal crossing angles from the non-crossing case.

**Multi-level metrics.** Some metrics measure geometric separability, namely how compactly leaf nodes lie within or are spread between metanodes, including edge length variability [130, 173], which is also applicable to single-level layouts. However, they do not recognize salience of clutter in a multi-level structure. The many topological separability metrics, such as modularity, conductance, coverage, performance, and so on [5, 47] do not measure geometric configurations. The only true geometric metrics in the literature that address the multi-level problem are ambiguity measures for community structure from Wang et al. [179] and the group overlap metric from Dunne et al. [39]. Wang’s approach focuses on the ambiguity caused by overlap of metanodes rather than readability, where they considered the geometric distribution of leaf nodes within metanodes rather than salience of clutter. It is later used in their follow-up papers to assess readability of clusters [70, 180]. Dunne’s proposal for group overlap does compare overlaps between a metanode and leaf nodes, but not between metanodes, which are more visually salient. Our
approach features a uniform treatment of nodes and metanodes to capture the clutter in both single-level and multi-level graph layouts.

**Sprawl metrics.** Several simple sprawl-related metrics have been proposed, including total drawing area, the maximum edge length, sum of edge lengths, and the aspect ratio of the drawing canvas [36]. The multiplicative inverse of sprawl, the metric that we use, has previously been called compactness by Kieffer et al. [92] and visualization coverage by Dunne et al. [39]. We use the inverse quantity to emphasize the problematic nature of sparsity as analogous to clutter. Kieffer et al. found evidence of its importance in small user-generated graph layouts [92], and Dunne et al. proposed it as a stand-alone separate metric for NN. The novelty of our work lies in exploring the trade-off between sprawl and clutter, not introducing the sprawl metric itself.

**Compound metrics**

Individual single-purpose metrics can be combined into a compound metric. Such compound metrics are often implicitly incorporated into automatic layout algorithms, as with physics models that measure a graph layout with an artificial physics quantity, such as force (GEM [52]), energy (LinLog [130]), or stress (NEATO [53]), as if the nodes and edges are real-world physical objects. For example, force models typically encourage uniform edge lengths and clustering around high-degree nodes, and discourage node-node overlaps. Some proposals have been made to add other types of forces to explicitly incorporate more single-purpose metrics [37, 84]. A compound metric explicitly proposed in previous work is a weighted sum of multiple single-purpose metrics [85], but justifying the weights remains an open problem. Our sprawlter metric, where sprawl and clutter is combined, is also an example of an explicit compound metric. We emphasize the specific trade-off between clutter and sprawl as directly opposite quantities, rather than attempting to combine very disparate quantities of all possible metrics into a single compound quantity.
3.2.2 Evaluation of graph drawing metrics

Previously proposed metrics have been evaluated either with human subjects or mathematical models, with the goal of obtaining a deeper understanding of the nuances and ranking the metrics by perceptual or cognitive impact. One approach is to gather quantitative performance data or qualitative feedback on the stimuli graph layouts from human observers, where the stimuli can be generated by automatic computational algorithms or human manual labour. There are many papers from earlier years that followed this general approach. Typical examples are from Purchase et al. [140, 142–144], Ware et al. [182], Huang et al. [82, 83], Kobourov et al. [95]. Another approach is to gather user-generated layouts in order to understand their mental model of graph layouts. Van Ham and Rogowitz asked participants to position the nodes in a graph layout and investigated which metrics are useful for explaining their layouts [173]. Following this thread, Dwyer et al. compared user-generated layouts with automatic layouts [44]. A third approach focuses on quantitative models of layouts and relevant tasks. McGuffin et al. evaluated space efficiency of visual representations of trees mathematically and provided design guidelines accordingly [118]. Dawson et al. proposed and evaluated a predictive model of human behaviour tracing paths through a graph [32]. None of these evaluations provide direct guidance on how to address the three problems that we tackle in this work.

3.3 sprawlter metrics

We describe the computation of the sprawlter metrics in this section including the general idea, technical details, and how they address the three problems presented in the introduction. First, for each clutter event, the amount of overlap (area of node-node overlap, length of node-edge overlaps, and angle of edge-edge crossing) is measured. Then, the overlap measurement is transitioned into a penalty – an indication of the degree of clutter – with a penalty mapping function. We elaborate further on these penalty mapping functions in Section 3.4. Finally, these penalties for each individual clutter event are summed up for node-node overlap (NN), node-
edge overlap (NE), and edge-edge crossing (EE) respectively as the area-aware metrics. Meanwhile, sprawl is also computed based on geometric properties of the entire layout. The sprawlter metric is a geometric mean combining the sprawl and area-aware metric.

3.3.1 Graph layout definitions

We define a graph \( G = (V, E) \), where \( V \) is the set of nodes and \( E \) is the set of edges. The graph \( G \) can be either a single-level graph or a multi-level graph; in the latter case, we simply expand the definition of the nodes \( v \in V \) to be either leaf nodes or metanodes. Since we treat both of these cases in a completely uniform way, our algorithm does not need to further consider the hierarchy \( H \) that is part of a compound graph, except for a low-level implementation detail (when avoiding computation of overlaps between a metanode and its descendants). We do not consider metaedges, which are analogous to metanodes, but are not commonly used in current layout algorithms.

We denote the geometric measurements by functions of graph elements: the area of a node \( \text{area}(v) \), overlapping area of two nodes \( \text{area}(v_1, v_2) \), length of an edge \( \text{length}(e) \), diameter of a node \( \text{length}(v) \), and overlapping length of a node-edge pair \( \text{length}(v, e) \). The crossing acute angle (in radians) of two edges is \( \text{angle}(e_1, e_2) \), and its complement is \( c\_\text{angle}(e_1, e_2) = \pi/2 - \text{angle}(e_1, e_2) \). When two graph elements just begin to touch, we consider them to intersect but with zero overlap: \( \text{area}(v_1, v_2) = 0 \), or \( \text{length}(v, e) = 0 \). When they do not intersect, the measurement function is undefined – it is not zero.

3.3.2 Measuring geometric overlap

We measure the overlapping area of a node pair, overlapping length of a node-edge pair, and (the complement of) crossing angle of an edge pair if the nodes or edges intersect with each other. Figure 3.2 shows an increasing amount of overlap between pairs from left to right, from none, to near-minimum, some, and near-
maximum overlap. Obviously, the degree of clutter corresponds to the amount of overlap. In the area case, the degree of clutter corresponds positively to \( \text{area}(v_1, v_2) \) or \( \text{length}(v, e) \). We consider length to be the degenerate version of area, and below often use \( \text{area} \) to mean both quantities. In the angle case, the degree of clutter corresponds positively to \( c_{\text{angle}}(e_1, e_2) \), where a perpendicular angle of 90 degrees is the least cluttered and a glancing angle of 0 degree – where one line segment is on top of the other – is the most cluttered. We thus consider the complement of the crossing angle rather than the crossing angle itself. In Figure 3.2, the count-based metrics are only able to distinguish between the leftmost column with no overlap and the other three columns, but not between those three.

![Figure 3.2: Increasing amount of overlap between node-node, node-edge, and edge-edge pairs. Count-based metrics can only distinguish no overlap from the other three cases, but not between them.](image)

### 3.3.3 Mapping measurements to penalties

We map the measured amount of overlap to penalties with a penalty mapping function in order to match the measurement to the degree of perceived clutter. Generally, we denote the penalty mapping functions by \( f(x) \), where \( x \) is the overlapping measurement. Specifically, for the three different families, we use \( f^{\text{NN}}(x) \), \( f^{\text{NE}}(x) \), and \( f^{\text{EE}}(x) \). Note that for the EE family, \( x \) denotes the complementary angle so that the valence of \( f^{\text{EE}}(x) \) matches the others (since an original angle of 0 is the worst case).
We avoid the trivial instantiation of \( f(x) \) where geometric measurements are used directly as penalties (i.e. \( f(x) = x \)), because the measured area / length / angle does not in general equal the degree of visual clutter. In particular, the distinction between the no overlap and touching cases is crucial (first and second columns in Figure 3.2), but \( f(0) = 0 \) would fail. The penalty mapping function is discussed further in Section 3.4.

### 3.3.4 Summing penalties

We directly sum up penalties of all overlapping pairs of node-node, node-edge, and edge-edge in a graph layout respectively, and refer to the three resulting total penalties as area-aware metrics, denoted by \( A^{NN} \), \( A^{NE} \), and \( A^{EE} \), while the count-based metrics are denoted by \( C^{NN} \), \( C^{NE} \), and \( C^{EE} \).

These area-aware metrics address the integer-count problem with a more precise indication of the degree of clutter. They also address the single-level problem for NN and NE, because the area awareness automatically captures the hierarchical information in multi-level graph layouts. Metanodes increase in area and diameter at ascending levels of the hierarchy: higher-level metanodes are larger than their constituent metanodes at lower levels, and than their leaf nodes at the lowest level. Hence, summing area or length penalties is analogous to a weighted sum of binary counts, where larger overlaps are weighted more heavily than smaller ones.

We show the pseudocode to compute the node-node area-aware metric in Algorithm 4; the similar analogs for NE and EE are in Supplemental Section B.2. The condition statement in line 1 checks for validity of the node pair, where the function \( \text{IsAncOrDesc}(v_1, v_2) \) checks if one node is the ancestor / descendant of the other in the node hierarchy. Function \( \text{CheckIntersection}(v_1, v_2) \) returns true if \( v_1 \) and \( v_2 \) intersect each other.
Algorithm 4: Computation of node-node area-aware metrics.

**Input:** \( G = (V, E) \)

**Output:** total penalty \( A_{NN}(G) \) and count \( C_{NN}(G) \)

1. totalPenalty ← 0
2. count ← 0
3. for \( v_1 \in V \) do
4.   for \( v_2 \in V \) do
5.     if \( v_1 \neq v_2 \) && \( \text{IsAncOrDesc}(v_1, v_2) \) then
6.       if \( \text{CheckIntersection}(v_1, v_2) \) then
7.         \( x \leftarrow \text{ComputeOverlapArea}(v_1, v_2) \)
8.         penalty ← \( \text{PenaltyFunc}(x, v_1, v_2) \)
9.         totalPenalty ← totalPenalty + penalty
10.        count ← count + 1
11. return totalPenalty, count

3.3.5 Combining clutter metrics with sprawl

While clutter indicates overlaps and occlusions of nodes and edges, sprawl indicates geometric sparsity of graph elements on a drawing canvas, as illustrated in Figure 3.1. The relevant factors contributing to sprawl include total drawing area (typically the axis-aligned minimum bounding box of the entire layout), node areas, edge lengths, and the number of nodes and edges. Spreading out nodes of a fixed size on a larger canvas would increase the sprawl, for example.

We quantify sprawl by dividing the total drawing area of a graph layout \( G \) by the area occupied by all nodes (excluding overlaps). It is the reciprocal of the compactness from Kieffer et al. [92] or the visualization coverage from Dunne et al. [39]. Formally:

\[
S(G) = \frac{\text{area}(G)}{\text{area}(\bigcup v)}
\]  

(3.1)

\( S(G) \) denotes the sprawl, and \( \text{area}(G) \) the total drawing area of the graph layout.
The area of union of all nodes captures the intuition behind sprawl well by accounting for variation in node sizes. We also considered other alternatives. If we use the number of nodes, $|V|$, as the denominator, it would not account for absolute node sizes: for example, the sprawl of layout (b) and layout (c) in Figure 3.1 would be the same. If we use the smallest node area, $\min_{v \in V} \{area(v)\}$, the resulting ratio would not gracefully handle relative node size variation: for example, a graph with 1000 large nodes and one tiny node would be over-penalized.

There is a direct trade-off between clutter and sprawl: for a fixed node size, the more spread-out the nodes are, the less clutter there would be and the more sprawl there would be. However, the interplay between clutter and sprawl is not strictly a zero-sum game: it is very possible to keep clutter low while having perceivable node sizes (or edge lengths) and reasonable total drawing area. It is common to combine multiple values by computing a mean of them. In mathematics, there exists many different versions of means [21] for different purposes. We choose the geometric mean (i.e. square root of the product of two positive values) because it is capable of normalizing different ranges of the constituent values: e.g. a 10% change in sprawl has the same effect as a 10% change in clutter. It is more suitable than the commonly-used alternatives of the harmonic mean, arithmetic mean, and quadratic mean. The harmonic mean is usually dominated by the minimum of the two values; a typical usage is to combine precision and recall for machine learning models. The arithmetic mean requires that the values have identical range to avoid unfair averaging. The quadratic mean is often used to measure errors between estimations and ground truths.

The sprawlter metric (denoted by $T$) for NN is stated as follows, and those of NE and EE have the same form.

$$T^{NN}(G) = \sqrt{S(G) \cdot \max\{A^{NN}(G), 1\}}$$ (3.2)

We impose a lower bound for the area-aware component: when the area-aware metric, or generally, the clutter measurement, is very small, the sprawlter metric should retain the sprawl-aware information; otherwise sprawl has little influence
on the sprawlter metric in such situations. In contrast, we do not need to impose any bound for the sprawl as it is always greater than or equal to 1.

We note that the general idea of sprawlter is completely independent of area-awareness: an instantiation could be designed for purely count-based metrics.

### 3.4 Penalty mapping function

The previous section provides an overview of the sprawlter metrics computation, which has the penalty mapping function at its core. We now discuss this crucial and challenging component in detail.

The penalty mapping function, $f(x)$, takes a geometric measurement of a single overlap event (area or length or complementary angle), and emits a number to indicate the degree of clutter for this event. In short, it is a mapping from overlap to clutter.

#### 3.4.1 General requirements

We identify four important general requirements for the penalty mapping functions, illustrated in Figure 3.3.

**R1. Increasing penalty.** $\forall x_1 < x_2, f(x_1) < f(x_2)$. The penalty should increase as the overlap increases. In Figure 3.3, $f(x)$ for the area-aware approach increases from a minimum to a maximum penalty, as opposed to a flat line for count-based approach. This requirement is necessary to distinguish between minimum, some, and maximum degree of clutter.

**R2. Substantial touching penalty.** $f(0) \gg 0$. In order to clearly distinguish the no-overlap case from a very small overlap, we require a substantial minimum penalty that is incurred as soon as two elements touch. That is, $f(0)$ should be substantially greater than zero, in contrast to $f(\text{undefined})$ that is exactly 0.
**R3. Count calibration.** $\text{minPenalty} < 1 < \text{maxPenalty}$. The sprawlter metrics are intended to be a drop-in replacement to count-based metrics but to provide richer information, and thus the penalties need to bracket, and be directly comparable to, the count-based value of 1. The touching case creates less clutter than what is captured by a simple count; the full overlap case demands a higher penalty.

**R4. Local function.** $f_{v_1,v_2}^{NN}(x)$ instead of $f_{G}^{NN}(x)$. We need to have a function that is calculated separately for each pair of nodes that can incorporate local knowledge of their individual sizes, rather than an identical global function that applies to all node pairs independently of their size. The EE family is a special case compared to the NN and NE one. Since all crossing angles share a universal unit (radians) and are bounded within the range $[0, \pi/2]$, a local function is equivalent to a global one. In the notation, we use subscripts to indicate whether the function is local or global.

In order to calibrate penalties to counts (R3), it is necessary to normalize the absolute node areas (and edge lengths) to relative areas. We divide the area of each node by the smallest node area in the layout such that the normalized area of the smallest nodes is 1. If all nodes have the same size, the normalized area of every node is 1; if node sizes vary, the normalized areas of larger nodes are greater than 1.
Similarly, for lengths, we divide them by the diameter of the smallest node. There is no need to normalize angles as they are all in radians. We use normalized areas and lengths below, unless otherwise specified.

3.4.2 Specific choices

We make specific choices to fulfill the general requirements of the penalty mapping function, while there exist other alternatives. Our goal is to construct a specific function that is simple and easy to understand. Figure 3.4 shows the NN penalty mapping function for the smallest node pair.

C1. We choose a power function: $f(x)$ is linear to $x^\gamma$, where $\gamma$ denotes the exponent in the power function and dictates the curve shape of $f(x)$. In Figure 3.4, we show the curves for three different $\gamma$ values. We chose $\gamma$ based on previous empirical work [83, 162]. The longstanding psychophysical power law proposed by Stevens [162], which is a mapping between physical stimulus and perceived sensation, inspired our choice of $\gamma = 0.7$ for node-node overlap areas and $\gamma = 1.0$ for node-edge overlap lengths. More recently, Huang et al. [83] suggest that human performance is correlated to the crossing angle with a quadratic term in their preliminary user evaluation, so we use a quadratic function ($\gamma = 2.0$) for edge-edge crossing angles. To check the impact of this choice, we conducted theoretical and computational analysis comparing a linear EE penalty mapping function and a quadratic one. We found that difference between the two is very small except when there are many glancing crossing angles. The details of this comparison are in Supplemental Section B.1.2.

C2. We choose a minimum penalty proportional to the node sizes, specifically a fraction of the maximum possible clutter measurement $M$ that would be incurred if they completely overlapped: $f(0) = \alpha M^\gamma$, where $\alpha$ denotes the fraction, and $0 < \alpha < 1$. For example, for the smallest node pair $v_1, v_2$, the maximum overlap $M_{v_1,v_2} = 1$, and thus $0 < f_{v_1,v_2}^{NN}(0) = \alpha < 1$, which is compatible with the requirements of a substantial touching penalty (R2) and
calibration to count (R3). In a layout with variable node sizes, $M$ would
be different for node pairs of different sizes, resulting in a larger touching
penalty for large pairs and a smaller touching penalty for small pairs, so the
local function requirement is also met (R4). The touching penalty could be
greater than one for sufficiently large nodes. Similarly, the same amount of
overlap area or length ($x$) will have a bigger impact ($f(x)$) if it happens on
bigger nodes than smaller nodes.

C3. We choose a maximum penalty proportional to the node sizes, with $M$ as
above: $f(M) = \beta M^{\gamma}$, where $\beta$ denotes the proportion, and $\beta > 1$. For the
same example as above where $M = 1$, R3 and R4 compatibility also holds,
and $f(M) = \beta > 1$. Meanwhile, as $\beta > 1 > \alpha$, the maximum penalty is guar-
anteed to be greater than the minimum penalty; combined with the choice of
a power function (C1), $f(x)$ is an increasing penalty (R1).

C4. We choose to set the halfway point of measured clutter for the smallest pair
to a penalty of 1: $f_{v_1,v_2}^{NN}(0.5) = 1$, $f_{v,e}^{NE}(0.5) = 1$, $f_{v,e}^{EE} (\pi/4) = 1$; where $v_1,v_2$
is the smallest node pair so $M_{v_1,v_2} = \text{min}\{\text{area}(v_1),\text{area}(v_2)\} = 1$, and $v,e$
is the smallest node-edge pair so $M_{v,e} = 1$. Note that the flat line for count-
based metrics lies exactly in the middle of minimum and maximum penalty
when $\gamma = 1$, as shown in the left function in Figure 3.4, but this statement
does not hold when $\gamma \neq 1$. The choice of halfway point is a specific way
of calibrating penalties to counts (R3). It also constrains the relationship
between $\alpha$ and $\beta$ so that only one of them needs to be specified beforehand,
improving the simplicity of the function.

### 3.4.3 Formal function instantiations

Combining the general requirements and our specific choices, we can define the
penalty mapping function in a general form:

$$f(x) = (\beta - \alpha)x^{\gamma} + \alpha M^{\gamma} \quad (0 \leq x \leq M) \quad (3.3)$$
This general form comes with three parameters, $\alpha$, $\beta$, and $\gamma$, which are responsible for minimum penalty, maximum penalty, and curve shape respectively. Our choice of $\gamma$ values was guided by empirical results from previous work [83, 162], as discussed in C1 above. With the halfway point choice (C4), i.e. $f(.5) = 1$, we are able to remove $\beta$ from the general form. We can also constrain $\alpha$ using the choices for minimum (C2) and maximum (C3).

\[
\begin{align*}
(\text{NN}) & \quad \beta = (1 - \frac{1}{5.7})\alpha + \frac{1}{5.7} \approx 1.625 - .625\alpha \quad (0 < \alpha < 1) \\
(\text{NE}) & \quad \beta = 2 - \alpha \quad (0 < \alpha < 1) \\
(\text{EE}) & \quad \beta = \frac{16}{\pi^2} - 3\alpha \quad (0 < \alpha < \frac{16-2\pi}{3\pi^2} \approx .328)
\end{align*}
\]

Inserting the known parameters $\beta$ and $\gamma$, we obtain the specific instantiations of penalty mapping function for NN, NE, and EE.

---

**Figure 3.4:** Illustration of the penalty mapping function for node-node overlap of a pair of the smallest nodes ($M_{v_1,v_2} = 1$) in the graph layout. The three functions show three different choices of curve shape ($\gamma$).
\[ f_{NN}^{V_1,V_2}(x) = (1 - \alpha)(2x)^{0.7} + \alpha M_{v_1,v_2}^{0.7} \quad (0 \leq x \leq M_{v_1,v_2}) \] (3.5)

\[ f_{NE}^{V_1,V_2}(x) = 2(1 - \alpha)x + \alpha M_{v_1,v_2} \quad (0 \leq x \leq M_{v_1,v_2}) \] (3.6)

\[ f_{EE}^{G}(x) = \left( \frac{16}{\pi^2} - 4\alpha \right)x^2 + \alpha \frac{\pi^2}{4} \quad (0 \leq x \leq \frac{\pi}{2}) \] (3.7)

### 3.4.4 Parameter space analysis

We conducted a mathematical analysis to understand the influence of the unspecified parameter in the formal statement above, namely, the minimum penalty fraction \( \alpha \). In addition to the theoretical analysis presented here, we also report on a computational analysis that allows us to suggest practical choices for values in Supplemental Section B.1.1.

We only report on the analysis of \( \alpha \) in the NN penalty mapping function, since \( \alpha \) for NE and EE functions is very similar. Figure 3.5 shows the NN penalty mapping function for the smallest node pair \((M_{v_1,v_2} = 1)\), using 6 different \( \alpha \) values within the range \((0, 1)\): 0.01, 0.2, 0.4, 0.6, 0.8, 0.99. We also include the function plot for a node pair that is 10x larger \((M_{v_1,v_2} = 10)\) in Supplemental Figure B.1, where the shapes and relationships between curves remain the same except that the x and y values are 10x larger.

First, we observe that as \( \alpha \) increases from 0 to 1, the range of \( f(x) \) decreases, and when \( \alpha \) takes the extreme value 0.99, \( f(x) \) is approximately a flat line, degenerating to the count-based metrics case. Mathematically, the range of the penalty mapping function is \( f(M) - f(0) = (\beta - \alpha)M^\gamma = (1 - \alpha)(2M)^{0.7} \). Semantically, the range represents its ability to distinguish between minimum, some, and maximum overlap. This ability has negative correlation with \( \alpha \) according to both our observation in Figure 3.5 and the mathematical representation above. Second, we observe that as \( \alpha \) increases from 0 to 1, the touching penalty \( f(0) \) (i.e. the y-intercept of the curves) also increases from 0 to 1 in Figure 3.5. Mathematically,
Figure 3.5: The node-node overlap penalty mapping function of the smallest node pair ($M = 1$), with different minimum penalty fraction ($\alpha$) values. The $\alpha$ controls the minimum penalty (y-intercept) as well as the difference between maximum and minimum (function range).

The touching penalty is $f(0) = \alpha M^\gamma = \alpha M^{0.7}$. Semantically, the touching penalty represents its ability to distinguish no overlap from minimum overlap, which has positive correlation with $\alpha$. Third, the changes of the range and minimum of the function is linear to the change of $\alpha$.

Therefore, $\alpha$ is the trade-off between the ability to distinguish different amounts of overlap and the ability to distinguish no overlap from touching.

### 3.5 Data generation

To validate the sprawlter metrics, we created small synthetic graphs with both manual and force-directed layouts, and applied four different layout algorithms to large real-world graphs, resulting in 56 layouts in total: 38 synthetic and 18 real-world layouts.

We created 9 small synthetic graphs, with under 50 nodes and 150 edges. We either manually positioned the nodes or applied off-the-shelf force-directed layout algorithms in Tulip [10] to make 38 layouts for five targeted purposes: problem
demonstration, debugging and sanity check, progression of clutter, variable sprawl, and limitations of our metrics. We varied the amount of clutter between NN, NE, and EE pairs, and the overall layout sparsity by pushing nodes away from each other or shrinking node sizes.

To confirm that the sprawlter metrics work with large real-world graph layouts, we used 4 large real-world graphs with 100-5K nodes and 500-10K edges. These are an academic coauthor network benchmarked in the Koala layout algorithm paper [87], a different coauthor network from the InfoVis 2004 contest [51] benchmarked in the GrouseFlocks layout paper [7], an email network provided by Leskovec et al. [102], and the add32 benchmark graph from the graph partitioning archive [160]. Supplemental Table B.1 provides a complete list. We applied the following layout algorithms to the real-world graphs: two single-level layouts available within Tulip [10], namely GEM [52] and FME [62] (the single-level version of FM3 [68]), the two-level Koala layout designed to emphasize cluster structure [87], and the GrouseFlocks multi-level layout [7]. We chose these based on implementation availability, computation speed, and ability to handle multi-level structure.

Our full computational pipeline is documented in detail in Supplemental Section B.5. In brief, we convert a variety of input graphs into the Tulip format, apply layout algorithms, extract geometries and the node hierarchy, compute the metrics, and finally display and analyze results for comparison. Except for applying the layout algorithms, our code is written in Python. In the supplemental materials, we include all input files, output files, implementation code, and Python scripts for results analysis. Supplemental materials are available at http://www.cs.ubc.ca/labs/imager/tr/2020/sprawlter/sprawlter-supplemental.zip and open source code is available at https://github.com/zipengliu/sprawlter-metrics.

3.6 Results

We conduct a comparative analysis between the sprawlter metrics, count-based metrics, and two from Dunne et al. [39], with respect to qualitative assessment of layout pictures. We report on the analysis approach, and then discuss instructive
examples across all three of the NN, NE, and EE families. For computational time, we show that our approach only costs a constant multiplicative factor to the count-based approach. Finally, we present the computational parameter analysis for the penalty mapping functions, guided by the theoretical analysis in Section 3.4.4.

A large table with pictures and metric values for all 56 layouts is available in the supplemental materials. Their running times are shown in Table B.2.

3.6.1 Analysis approach

We compute the area-aware clutter penalties ($A$), sprawl ($S$), sprawlter ($T$), and counts ($C$) for each graph layout, in order to validate them separately. We also derive the average penalty $A/C$, the ratio between the penalties and counts, to help us find representative cases. When the average penalty is below one ($A/C < 1$), the overlap areas, lengths or crossing angles are less cluttered than the count suggests, which is typically caused by overlapping between small nodes and edges, near-touching overlaps, or near-orthogonal crossings. When the average penalty is above one ($A/C > 1$), the overlaps are more cluttered than the count suggests, which is typically caused by overlapping between big nodes and edges, nearly full overlaps, or glancing angle crossings.

We also compute the two most relevant metrics proposed by Dunne et al. [39] (denoted by $D$), the global node-node overlap and the global crossing angle. The node-node overlap metric is the ratio of the area of the union of the node representations over their total area if drawn independently (perceived readability over best readability). A point not discussed explicitly in their paper is that this union formula assumes nodes are at the same level. It is not obvious how to extend it to multi-level layouts, since the union of parent metanodes and their constituent children would be an over-count. We thus compute it only on leaf nodes. The edge-edge crossing angle metric computes deviations from a specific optimal angle, a parameter that they set to $70^\circ$ according to the empirical study from Huang et al. [83]. Although Dunne et al. also propose a group overlap metric that involves metanodes, it only counts the crossings between metanodes and leaf nodes, but not
between metanodes, so we do not compute it.

We compare the five computational results \((A, S, T, C, D)\) to the layout pictures, and make a subjective qualitative assessment of correspondence between the metrics and quality of layout. For the first four metrics, low numbers are good and high numbers are bad. For \(D\), 0 means the worst and 1 the best.

### 3.6.2 Comparative analysis

We present several representative cases in different types of graphs and layout algorithms, and discuss how the tested metrics reflect the readability of these cases. (We use \(\alpha = 0.20\) for NN, NE, and EE in this section.)

**Node-node overlap**

**Increasing metanode overlaps (synthetic).** Figure 3.6 shows a small synthetic graph layout with increasing overlap between the red and blue metanodes (from left to right). The area-aware metrics for NN \((A^{NN})\) also increase \((4.27 < 14.36 < 30.62)\), but the counts \((C^{NN})\) do not \((1 = 1 < 12)\) and neither does Dunne’s NN metric \((1.00 = 1.00 > 0.90)\). Looking closer at the breakdown by metanodes and nodes, the penalties of metanodes \((4.27, 14.36, 22.05)\), compared to penalties of leaf nodes \((0, 0, 11)\), are the major contribution towards the totals, which matches our intuition that big metanode overlap should be penalized more heavily than that of leaf nodes, whereas the breakdown of counts does not match this intuition.

**Force-directed layouts for variable-size nodes (synthetic)** Figure 3.7 shows a single-level graph using two force directed layout algorithms, namely Davidson-Harel [31] and stress majorization [53]. The nodes have variable node sizes, as can occur either where node size is mapped to some quantitative attribute, or where nodes contain variable-length labels. We can see that there are no overlaps in the right layout and a few overlaps between large nodes in the left layout \((16.05 > 0)\), but the right one is much more sparse \((1605.28 >> 6.09)\). Overall, the left has
Figure 3.6: Increasing overlap between red and blue metanodes in a small 2-level synthetic graph. Only closeups for the overlaps are shown. A and C are broken down by penalties / count of metanodes and leaf nodes. The area-aware approach (A) matches with the visible configuration better than count-based (C) and Dunne’s (D) approach.

<table>
<thead>
<tr>
<th></th>
<th>Near-min</th>
<th>Some</th>
<th>Near-max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4.27+0=</td>
<td>4.27</td>
<td>14.36+0=</td>
</tr>
<tr>
<td>C</td>
<td>1+0=</td>
<td>1</td>
<td>1+0=</td>
</tr>
<tr>
<td>D</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

Many touching leaf node pairs and high sprawl (real-world). Figure 3.8 shows a large real-world graph from the partition benchmark dataset, with 4960 nodes and 9462 edges, using FM3 layout (single level). The most noticeable feature about this layout is that the nodes are too tiny to be seen and there is huge wasted space on the upper left and bottom right, resulting in a large sprawl (319.28). Despite the many touching overlaps between the tiny leaf nodes, as shown in the inset on the upper left, the area-aware penalty is still less than count (213.28 < 670), showing that the penalties for these small overlaps sum to an appropriate amount that is non-zero but not disproportionate. Dunne’s metric (almost 1.00) does not reflect the wasted
Node-node overlap and sprawl

Davidson-Harel                          Stress majorization

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>16.25</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>6</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>6.09</td>
<td>1,605.28</td>
<td>40.07</td>
</tr>
<tr>
<td>T</td>
<td>9.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0.91</td>
<td></td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Figure 3.7:** A single-level synthetic graph with variable node sizes. The left layout has a lower sprawlter metric than the right layout although there are substantial overlaps between nodes; we argue that qualitatively the readability is indeed better.

---

**High vs low sprawl (real-world).** Figure 3.9 shows an email network layout by GrouseFlocks, with only a few metanodes open and others closed. There are only 61 nodes and more than 1000 edges in the left layout, and 88 nodes and more than 2000 edges in the right layout. As the overlaps only happen between metanodes, Dunne’s metric (1.00) is not helpful. The left layout is less cluttered than the right one in terms of both area-aware penalties (133.52 < 184.13) and counts (16 < 40), but the left is less efficient in space usage than the right (i.e. higher sprawl, 947.59 > 219.59), resulting in a higher sprawlter (335.71 > 201.08). This example and the previous one suggest that both clutter and sprawl are important aspects of measuring graph layout quality, and the trade-off between the two should be considered.
Figure 3.8: Large real-world graph layout by FM3 in Tulip (single-level), with high sprawl and many touching events. The average penalty ($A/C$) indicates penalties for overlaps are small but non-zero, and the high sprawlter value ($T$) indicates that this layout is bad in terms of sprawl ($S$) and clutter ($A$).

Node-edge overlap

Increasing metanode and edge overlaps (synthetic). Figure 3.10 shows a progression of node-edge overlap on a small synthetic layout. Both the area-aware metrics ($12.11 < 54.46 < 136.72$) and counts ($6 < 15 < 36$) are increasing to indicate the increasing overlap between the blue metanode and edges, and also the constituent leaf nodes and edges, but the increasing average penalty ($2.02 < 3.63 < 3.80$) indicates that the penalty increases at a faster pace. As in the analysis of Figure 3.6, the breakdown of penalties by metanodes and leaf nodes shows that overlap of metanodes and edges contributes more than that of the leaf nodes. The area-aware metric better aligns with our judgement of the layout pictures, because
Figure 3.9: Two large multi-level real-world graph layouts by GrouseFlocks (4-level). The left layout has higher sprawl but less overlap than the right one, captured by the sprawl $S$, and clutter with area-aware approach ($A$) and count-based one ($C$).

the clutter between nodes and edges deteriorates as the blue metanode moves upwards.

**Increasing metanode and edge overlaps (real-world).** Figure 3.11 shows the comparison between area-aware metrics and count-based metrics on a real-world example. On the left layout, all leaf nodes are hidden; on the right layout, we open up two of the orange metanodes, making it more cluttered. For the node-edge overlap, there are significantly fewer overlaps on the left layout, which is reflected by both metrics ($4482.20 < 32618.75$, $539 < 1650$), and by the average penalty ($8.32 < 19.77$). Notice that there are many edges that fully cross the biggest green metanode on the right layout, resulting in an extremely high average penalty. Also, the fact that $A >> C$ for both layouts matches our intuition that overlaps related to
### Node-edge overlap

<table>
<thead>
<tr>
<th></th>
<th>Near-min</th>
<th>Some</th>
<th>Near-max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5+7=</td>
<td>40+15=</td>
<td>98+39=</td>
</tr>
<tr>
<td></td>
<td>12.11</td>
<td>54.46</td>
<td>136.72</td>
</tr>
<tr>
<td>C</td>
<td>1+5=</td>
<td>5+10=</td>
<td>8+28=</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>15</td>
<td>36</td>
</tr>
<tr>
<td>A/C</td>
<td>2.02</td>
<td>3.63</td>
<td>3.80</td>
</tr>
</tbody>
</table>

**Figure 3.10:** Increasing overlap between blue metanodes and edges in a small 2-level synthetic graph. Although counts (C) are increasing, the area-aware approach (A) captures the deteriorating situation in the picture more precisely, visible via the increasing ratio of average penalty (A/C).

Big metanodes are visually salient and should be penalized heavily.

### Edge-edge crossing

**Decreasing crossing angles (synthetic).** Figure 3.12 shows a progression of decreasing angles between crossing edges on a small single-level synthetic graph, from orthogonal to near-glancing angles. There are exactly two edge-edge crossings on each layout, but the area-aware metric is increasing (0.99 < 1.46 < 3.45 < 4.42). In two layouts on the right, the edges are almost on top of each other, which looks a lot worse than the two layouts on the left. The count does not indicate such clutter on this naive example, whereas the area-aware metric is a better match to the degree of clutter of the edge-edge crossings. Dunne’s metric also goes down as clutter increases (0.73, 0.87 > 0.28 > 0.10) (least clutter is reached at 70°), but reports near-maximum EE readability in the orthogonal case which ignores the existence of remaining edge-edge crossings.

**Orthogonal vs glancing angles (real-world).** Figure 3.11 shows similar results.
Figure 3.11: Two large multi-level real-world graph layouts by GrouseFlocks (4-level). The area-aware NE metric ($A^{NE}$) captures the property that there is less node-edge overlap on the left than the right. The area-aware EE metric ($A^{EE}$) captures that on the left the crossing angles between edges are closer to orthogonal angles, while on the right they are closer to glancing angles.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td></td>
<td>4,482.20</td>
<td>32,618.75</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>539</td>
<td>1650</td>
</tr>
<tr>
<td></td>
<td>A/C</td>
<td>8.32</td>
<td>19.77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EE</td>
<td></td>
<td>703.67</td>
<td>5,936.89</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>808</td>
<td>4,154</td>
</tr>
<tr>
<td></td>
<td>A/C</td>
<td>0.87</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>0.73</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Figure 3.12: Decreasing crossing angles between edges in a small single-level synthetic graph, captured by the area-aware EE metric ($A^{EE}$).

<table>
<thead>
<tr>
<th>Edge-edge crossing</th>
<th>Orthogonal</th>
<th>Near-half</th>
<th>Small</th>
<th>Near-glancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.99</td>
<td>1.46</td>
<td>3.45</td>
<td>4.42</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0.73</td>
<td>0.87</td>
<td>0.28</td>
<td>0.10</td>
</tr>
</tbody>
</table>

on real-world examples. From the layout pictures, we can see that the edge-edge crossings on the left layout are mostly near-orthogonal, while those on the right
layout are mostly small and even near-glancing angles. Not only are there fewer edge-edge crossings on the left (808 < 4154), but the clutter caused by crossing angles is also less, which is reflected by the average penalties for EE (0.87 < 1.43). In this case, Dunne’s metric (0.73 > 0.46) is also successful in differentiating the two layouts.

### 3.6.3 Computational time

As we see from algorithm 4, the sprawlter metrics are computed along with the counts with a computational complexity linear in the number of NN, NE, and EE pairs. Therefore, we expect that the computation of the sprawlter metrics would be in the same scale as that of the count-based metrics, with a constant factor slowdown.

For every graph layout measured, we timed the full computation of the sprawlter metrics, and also of the count-based metrics by skipping the code only relevant to sprawlter metrics; that is, operations within the inner condition in algorithm 4 are all skipped except the one that increments count (line #10). We used a 2012 MacBook Pro with a 2.5 GHz Intel Core i5 CPU and 8GB RAM to run all computations, and report the average computational times over 4 runs. The absolute running time is related to the number of nodes and edges, and also the number of crossings. For example, for a large real-world graph with about 5k nodes and 10k edges (Figure 3.8), it takes 128s, 505s, and 510s to compute the NN, NE, and EE sprawlter metrics respectively, while it takes 92s, 358s, and 372s to compute the counts.

We then compared the two execution times by deriving the ratio between running time of sprawlter metrics and counts, which is the equivalent of the slowdown factor. We excluded execution times under 1 second as unreliable estimates. The results align with our expectation. The average ratio over all included layouts is 1.38 for NN, 1.50 for NE, and 1.85 for EE, with relatively small standard deviations. Our findings suggest that the slowdown factor is a constant, which ranges from roughly 1 to 3.
Our Python implementation of the sprawlter metrics serves adequately as a proof-of-concept, but there are obvious directions for optimization. Our naive algorithm simply iterates over every pair of nodes (and edges in NE and EE case). It would be possible to apply space decomposition algorithms to speed up the computation, such as the Bentley-Ottmann line sweeping algorithm \[14\] to compute all crossing edges.

### 3.7 Discussion and Future Work

Overall, the area-aware approach clearly outperforms the simple count-based approach in the NN, NE, and EE cases, and also the Dunne approach for the NN case. Our results are somewhat better than Dunne’s EE case: more edges do imply more crossings, but not necessarily worse crossing angles. The area-aware approach is more complicated than the others due to a parameterized penalty mapping function; we argue that the benefit of precision is worth this cost.

Our approach does not impose an upper bound for the metrics, in contrast to Dunne’s metrics that are within \([0, 1]\). Their metrics are essentially ratios of the current readability and the best achievable readability, where 1 corresponds to the most readable and 0 to the least readable. In particular, they propose normalizing the EE metric by the \(O(|V|^2)\) term of all possible edge crossings in a complete graph, but this number is misleading because the density levels of node-link graph drawings must be far below that ceiling in order to be comprehensible; Melancon \[121\] argues that typically \(|E| < 4|V|\). While having a fixed bound is beneficial in comparing the same graph with different layouts, it is not for comparing different graphs. Their metrics are ratios relative to the graph itself, comparing the perceived readability against the best it could be, but the same number would have a very different meaning for a small and simple graph than for a large and complex graph.

We argue for the necessity of clutter metrics despite the existence of clutter removal algorithms \[42\]. The entire point of our area-aware clutter metrics is to more precisely quantify the clutter that results from overlap. It could be used algorithmically
to determine which layout algorithm to use for a particular graph; for example, whether the computational cost of overlap removal is worth the improvement in the layout quality. Moreover, overlap removal sometimes results in an increase of sprawl (especially for large graphs), so the trade-off between sprawl and clutter can help determine to what extent we want to remove overlap completely, partially, or just leave it. Moreover, many layout algorithms used in practice do not guarantee the total elimination of all overlaps; their approach to accelerating computation may leave some in place. We would like to see these new metrics incorporated into layout algorithms, which would be easy to do with optimization-based approaches.

One limitation of the sprawlter metric is that only clutter and sprawl are captured. Other readability factors that affect the overall quality of a layout include its ability to show cluster structure; using spatial proximity to show groupings may incur clutter costs as a direct trade-off, as in the Koala layouts (shown in the supplemental table of full results).

Our general area-aware approach could be extended to cover a wider range of metric families as future work. Another interesting direction of future work would be a resolution-aware approach. In the mathematical formulas, pixels can be divided into infinitely small sub-pixels. If two nodes just touch each other, the overlapping area is exactly zero. However, in the practical world, pixels on a screen are discrete. An explicit awareness of the display resolution would provide the precision required to exactly assess when there is no longer visible space between two items because they occupy neighboring pixels.

A very rich direction for future work would be to incorporate more aspects of multi-level structure by considering metaedges. Progress may not occur until more layout algorithms support them and more psychophysical experimentation has taken place to untangle the factors that affect edge-edge crossing clutter beyond the crossing angle.
3.8 Summary

We contribute the area-aware sprawlter metric for graph layout to capture important properties beyond integer counts of crossings on the leaf level by accounting for the amount of overlap and beyond clutter by accounting for the geometric sparseness of layout. We instantiate and implement the sprawlter metric, and present the technical details for the crucial component, the penalty mapping function. Through our validation with synthetic and real-world layouts, we show that the new metric succeeds at providing a better quality measure than previous work by taking into account both clutter and sprawl.
Chapter 4

LogSeg: Data-driven Multi-level Segmentation of Image Editing Logs

Automatic segmentation of logs for creativity tools such as image editing systems could support smart history navigation to improve their usability and learnability. We propose a multi-level segmentation model that works for many image editing tasks including poster creation, portrait retouching, and special effect creation. The lowest-level chunks of logged events are computed using a support vector machine model and higher-level chunks are built on top of these, at a level of granularity that can be customized for specific use cases. Our model takes into account features derived from four event attributes collected in realistically complex Photoshop sessions with expert users: command, timestamp, image content, and artwork layer. We present a detailed analysis of the relevance of each feature and evaluate the model using both quantitative performance metrics and qualitative analysis of sample sessions.

4.1 Motivation

Analyzing and modeling user interaction logs can address challenges posed by complex user interfaces in professional creativity tools. A user can issue many actions in a short period of time, but it could be challenging to keep track of the working process on a fully loaded user interface. Segmenting action logs into semantically meaningful pieces could alleviate one’s cognitive load of low-level information. A primary use case of segmentation is smart undo [26]: if we can automatically segment the log into chunks of actions, the user will be able to navigate editing history more easily by undoing or redoing a group of related actions instead of one action at a time. Other use cases that also rely on accurate identification of key decision or transition points in the log data include smart version control [25], automatic generation of tutorials from interaction logs, and recommendation of alternative design ideas or workflows [28].

In this chapter, we investigate the segmentation of image editing logs from Adobe Photoshop (PS). Figure 4.1 shows a concrete example of a segmented image log, introducing key terms used in this chapter. Designers and artists use PS for a variety of tasks, such as composing a poster, retouching a photo, and applying special effects to an image. A task is accomplished in a number of steps or subtasks, for example positioning images and tuning colors during poster creation. A subtask may be further decomposed into smaller subtasks, for example adding text involves making text content and applying text color. The hierarchy of subtasks in a complex PS task is not balanced; they may have different granularities. At the lowest level, a subtask is accomplished by a group of events, namely individual operations triggered through key strokes or mouse clicks. Each event has a number of attributes, including the command invoked, such as Move, and timestamp. The entire sequence of events logged in the process of accomplishing a PS task comprises a session.

We define segmentation as grouping consecutive events within a session into a hierarchy of semantically meaningful chunks. A perfect segmentation method will produce chunks that correspond to the subtasks. The desired subtask granularity
Session: a poster creation task

High-level chunks (subtasks)
- Position images
- Tune colors
- Make lighting around characters
- Add text
- Add more light to characters
- Check results

Low-level chunks
- Make title content
- Apply color to title
- 2nd trial using clipping mask
- Make a mask layer
- Adjust opacity

Events
- Type text
- Edit and move title
- Set title style
- Make subtitle
- 1st trial

Attributes

Figure 4.1: Concrete example of session segmentation. High-level chunks consist of low-level chunks, which consist of logged events, which have attributes. The single expanded chunk shows the non-balanced nature of the hierarchy.

depends on the downstream use cases: for example, smart undo and redo will require finer granularity, while creating an overview of the whole session can be more coarse-grained.

Despite the wide range of potential applications, many questions about segmenting image editing logs remain unanswered. First, image editing tools support diverse tasks such as photo retouching, image composition, special effects creation and user interface design. These different tasks result in very different workflows. However, existing segmentation methods usually are limited to one type of editing task [26, 192], and provide no evidence on whether segmentation models can generalize across task types.

Secondly, many segmentation approaches (e.g. hierarchical clustering [153], se-
quence progression models [186], infinite hierarchical hidden Markov models [74]) assume that command name and timestamp are the only two relevant event attributes. Other event attributes such as image content and editing area overlap have been included in segmentation models [25, 26]. The relevance of these event attributes for segmentation has not been thoroughly analyzed, and the suitability of other potentially relevant attributes has been ignored.

Finally, real-world user behaviors are complex due to individual differences. Collecting realistic workflow datasets with rich event attributes and labelling them as ground truth is expensive and time-consuming. As a result, the log datasets collected in previous work either enforce homogeneous workflows with predefined subtasks [26], or are unrealistically clean without user mistakes or digressions [25]. It is unclear if certain kinds of user behavior may pose challenges in creating unambiguous labels and building robust segmentation models.

In this chapter, we take a first step in addressing these problems, targeting smart undo [26] as a primary use case. Our goal is three-fold: build a reasonable model, understand the relevance of features, and investigate the performance of the model in the context of user interface applications given the complexity of heterogeneous tasks and individual variation in user behavior.

We collect log data of realistic workflows from PS expert users for different types of image editing tasks. For each event in a sequence, we record four attributes (command, layer, image content, and timestamp) that are potentially relevant for segmentation, and label chunk boundaries in all collected sessions as ground truth. We derive five features from these for use in machine learning (ML) models: command similarity, layer similarity, image diff, working region overlap, and duration. We then develop a two-stage segmentation model: a supervised machine learning model to predict the boundaries for low-level chunks, and two alternative approaches to construct high-level chunks from the low-level ones – multi-tier thresholds or agglomerative clustering. This two-stage scheme allows for the flexible choice of levels to support various downstream use cases.

Our primary contributions are: 1) a simple and interpretable low-level segmenta-
tion model that works reasonably well for smart undo, and 2) an analysis of feature relevance to segmentation, where we find that command, timestamp, and layer are relevant event attributes, but image content is not as useful. We are the first to use layers for segmentation and confirm their relevance. We validate the model with quantitative evidence showing that our model generalizes across three image editing tasks for low-level chunks. Our secondary contribution is an adjustable high-level segmentation model built on top of the low-level one. We conduct a preliminary qualitative analysis of the suitability of higher-level chunks for these three tasks. We also provide observations on the complexity of real-world user behavior and note the challenges it raises for multi-level segmentation.

4.2 Related Work

We briefly introduce the related work on event sequence visualization and graphical histories, then focus on computational models of event logs.

4.2.1 Event Sequence Visualization

Event sequences have been studied frequently in the visualization community for application domains including e-commerce, program execution traces, and medical records, where researchers focus on visualizing large-scale collections of event sequences to get data insights such as identifying common patterns or outliers [35, 65, 109]. In contrast, for segmenting image editing logs we are dealing with one sequence (session) at a time, so visualizing many sessions at once does not help with segmentation.

4.2.2 Workflows and Graphical Histories

Researchers have proposed graphical histories of workflows in domains ranging from general scientific analysis [22] to specific biological experiments [115] to the standalone visualization application Tableau [72, 114]. However, these domain-
specific approaches do not adequately address the interface problems of creativity tools such as image editing.

Work targeted at creativity tools addressed the question of how to represent and manipulate application state to undo and redo in single-user [46] and multi-user collaborative settings [156]. Other approaches used annotations on the graphical interface to inform user interaction history [127, 164], or supported the capture, exploration, and playback of creative document workflow histories [63]. The Delta system visually represented and compared a small amount of short workflows [96]. Grabler et al. [59] and Chi et al. [27] proposed an automatic generation of multimedia tutorials from logged sessions in GIMP (an alternative to Photoshop). The focus of these papers are either on data models or visualization of interaction history. Some of them tried to group interactions into chunks, as we do, but their goal was to support selective undo that ties to space on the canvas, whereas our goal is to group events into subtasks that might or might not relate to spatial proximity on the canvas.

4.2.3 Event Log Models

The previous work on segmentation models for general event logs using approaches such as dynamic programming [169], hidden Markov models [74], and probabilistic generative models [186] does not perform well in our setting of image editing logs with rich attributes. Moreover, some of them require large-scale data which we do not have. The work on non-segmentation models for general logs, such as bias detection in visual analytics logs [177], is more distant from our own; models targeted at creativity tool logs are more relevant.

Creativity Tool Logs

We take inspiration from many non-segmentation models of logs. Lafreniere et al. studied and characterized large-scale real-world image editing command logs [98], providing us with interesting patterns of command usage in image editing tools.
Adar et al. proposed CommandSpace to model the relationships between commands, subtasks, and text descriptions of subtasks [3], using machine learning models for natural language. Our command similarity feature is computed in a similar way. Two other approaches to learning usage semantics from command logs solidified our interest in ML approaches: Yang et al. proposed a deep learning model, using command logs, for representation of PS users [187], and Wang et al. proposed a two-step log classifier to recommend frequent patterns [178]. Causality, a conceptual model by Nancel and Cockburn [128], models causal relationships between events, which helps us think about structure within a session.

The few papers on segmentation models of creativity tool logs are most relevant to our work. Denning et al. proposed to use regular expressions to retrieve a multi-level segmentation interaction logs in a mesh construction application [33, 34], which does not apply in image editing case as there is more variation in our tasks. Chen et al. proposed a non-linear revision control system to model, aggregate, and visualize image editing logs [25]. However, the non-linear model is not intuitive for PS users, who can already achieve such non-linear edits with non-destructive editing techniques [135]. Moreover, their dataset seems unrealistically short and clean. Chen et al. later proposed to use a support vector machine to predict a single-level segmentation for portrait retouching logs [26], which specifically targets one task with predefined subtasks. They also compared to a baseline from Li et al. that uses duration for segmentation [107], and argued that duration is not effective. We were inspired by this approach to use support vector machines ourselves. Zhao et al. developed the Sketcholution system to get a high-level segmentation of sketching logs using an agglomerative algorithm [192], but it is not clear that the specific solution generalizes to other tasks. However, we were inspired to try agglomerative approaches as well. In this chapter, we are dealing with multiple common PS tasks instead of one, and we collect more realistic sessions than previous work. The use of layer as a feature is absent from all of this previous work, as is the systematic investigation of the relevance of each feature.
4.3 Approach

We provide an overview of our approach and its rationale for data generation, model selection and evaluation methods.

4.3.1 Data Generation

Based on literature review and our interaction with Photoshop users, we identify four potentially relevant event attributes: command, timestamp, image content, and layer. Command, timestamp and image content have been used in previous work [25, 26, 192]. Layer has not been considered in previous work, but we introduce it as a potentially important attribute because of our observation that layers are essential in any realistically complex PS tasks.

No existing logs captured all four of these attributes, so we created a logging PS plugin to record this information for each event during normal usage sessions. We conducted user studies to collect data from experts carrying out multiple tasks, capturing these logs and also think-aloud recordings. We used these records to manually create labels for chunks at multiple levels of granularity. For low-level chunks, we found good inter-coder agreement between two independent coders. However, for high-level chunks there was little agreement even between human coders. We decided that only the low-level chunk labels were reasonable to consider as ground truth, a decision that in turn affected our choices of models and evaluation strategies. We also concluded that deeper investigation of the relevance of segmentation attributes was merited.

4.3.2 Model and Feature Selection

The goal of the segmentation model is to predict whether an event $e$ is the starting point of a new chunk (similar to Chen et al. [26]). Considering that applications such as smart undo operate on streaming data during interactive sessions, we ruled out building models that required knowledge of both the events that precede and
those that succeed $e$. Instead, our model makes predictions based on only the events that precede $e$. We further choose to build on similarity (or distance) measures between events according to their attributes, under the assumption that similar events are likely to be in the same chunk.

Our finding that only the low-level chunk labels are reliable led us to develop a two-stage segmentation model. For the low-level chunks, we use a supervised approach built on the human labels as ground truth; we apply an unsupervised approach for segmenting higher-level chunks. We frame the low-level segmentation problem as an instance of binary classification in machine learning (ML), where the classes are 1 for the boundary and -1 for the non-boundary. We also conducted early experiments with multiple hand-designed rule-based models for low-level segmentation (e.g. segment when a user switches to a different layer), which we documented in Supplemental Section C.5, but found that they were substantially outperformed by a simple and popular ML model, a support vector machine (SVM). Since our focus is to gain a deeper understanding of the segmentation problem and we do not have large-scale data for training, we prioritize the interpretability of a simple model over more complex and powerful ones like random forests or neural networks. We use a linear kernel, again for simplicity. During development, we also tried SVM with a quadratic (RBF) kernel, but since it did not improve performance we stayed with the simpler approach.

The simple SVM model achieves good performance, as we show below in Results, with a small number of features (five) that we derive from the four logged attributes: command similarity, layer similarity, image diff, working region overlap, and duration.

### 4.3.3 Evaluation Methods

With reliable labels on low-level chunks, we are able to perform quantitative evaluation on the SVM model by analyzing feature relevance and the model’s performance on low-level segmentation. The lack of ground truth for the high-level segmentation led us to a qualitative-only evaluation: we constructed visualizations...
compact enough to show entire sessions with event-level detail, so that we could inspect the relationship between feature values and compare the predictions with human labels.

### 4.4 Data Collection and Characterization

We report our data collection procedure and characterize event attributes.

#### 4.4.1 Data Collection

Our study with expert users was conducted in two rounds of data collection, early and late in the project timeline. The data collected from the first round was used for exploratory analysis and model building, while data from the second round was mainly used for model validation and testing.

We compiled a pool of PS tasks (9 in round one and 11 in round two) from popular PS tutorial websites to use in the studies, falling into three types: poster creation, portrait retouching and special effects creation. We recruited 13 PS expert users with many years of in-depth professional experience (ranging from 2 to over 10 years), with 7 participants in each round; one person participated in both rounds. The study was run for each participant via online video conference, lasting around one hour in total. We first introduced the participant to the goal of the study, and helped them install the PS plugin. We showed them the pool of PS tasks, and asked them to choose one (or two) tasks that they felt most comfortable and competent working on. We provided all the required source images for the task, and a final image for reference (except for portrait retouching task). We asked the participants to think aloud throughout the session, and recorded the entire video conference with screen and audio information.

We collected a total of 16 sessions, 8 from each round, where one participant in each round did two sessions. Broken down by task type, there were 8 sessions for poster creation, 6 for portrait retouching, and 2 for special effect creation. There
were 5718 events logged in total across all sessions, with an average of 357 per session (min: 29, max: 1064), and each session lasted an average of 33 minutes (min: 8, max: 53). Our PS recording plugin saved command names, timestamps, and layer names & IDs in a text file, and the image content was saved as JPEG screenshots.

We manually segmented each session into multi-level chunks by labelling the starting and ending events of each chunk. We built an interface showing all the event attributes in a table and used it in conjunction with the video recordings to perform the labelling, as shown in Supplemental Figure C.9.

The labelling process involved subjective judgements about which events should be grouped as a chunk at the lowest level, and whether low-level chunks should be grouped into a higher-level chunks; for example, whether to group “Make title” and “Make subtitle” together in Figure 4.1. To assess inter-coder reliability, two coders independently labelled the starting events of lowest-level chunks in two sessions (a total of 351 events) and compared their results. We found that the two coders mostly agreed with each other (Cohen’s kappa $\kappa = 0.77$), which suggests that the low-level chunks are reliable. However, among the 14 incongruent events, 11 were caused by disagreement on granularity (e.g. one coder labeled a chunk of events 23-44, while the other labeled two chunks 23-39 and 40-44), indicating that there would be large inter-coder differences on high-level chunks. We thus chose not to compare human-generated high-level chunks between coders. This observation led to our decision to only treat the lowest-level labeled chunks as ground truth. One coder then continued to label all the remaining sessions.

### 4.4.2 Data Characterization

**Commands** are directives from users. The number of unique commands in PS is large: there are about 1400 menu items in Photoshop 2019, each corresponding to a command. Each command is associated with a meaningful and interpretable name, and is considered highly relevant to segmentation in previous work [25, 26].
Layers are essential for managing visual objects, achieving complex visual effects, and performing non-destructive editing. They are created, ordered, and possibly placed into groups within a hierarchy explicitly by the PS user; three examples are shown in Supplemental Figure C.4. During a session, users activate different layers at different points of time to work on, and the layer hierarchy evolves over time. The final count of layers in each session ranged from 3 to 16; the layer hierarchy was always shallow, with final depth ranging from 1 to 3. We consider layer a potentially relevant signal because a change in active layer could indicate a shift in subtask, and the changes in layer hierarchy reflect the user’s mental models of the artwork composition.

The image content of an event refers to the pixel state of the artwork after the operation has been carried out. The changes in image content after an event could be a useful indicator of the active working area, and may be relevant to segmentation.

The timestamp is used to derive duration. Duration between consecutive events could be relevant to segmentation, if rapid-fire events are more likely to occur within a chunk, whereas a long pause between two events might signal a boundary between chunks.

### 4.5 Feature Computation

We derive features for our segmentation model from the four event attributes collected in the expert user sessions: command, layer, image, and timestamp. The five features that we derived are specific instances of a similarity (or distance) measure between events: command similarity, layer similarity, image diff, working region overlap, and duration.

#### 4.5.1 Command Similarity

A conventional approach to quantify command similarity is to group commands according to the menu hierarchy in the Photoshop interface [26]. This approach
fails to differentiate commands within the same cluster, and might not capture how commands are used in reality. An alternative approach, validated in the CommandSpace previous work [3], is to learn the semantics of the words using machine learning tools for large natural language datasets. The rationale is that commands are analogous to words and a PS session is equivalent to a document, where the proximity of logged commands within a session expresses meaningful semantics that capture typical usage patterns. Inspired by this idea, we obtained access to a command log database internal to Adobe, which only recorded commands for PS events from opt-in non-enterprise users. We extracted 100 million commands from this database and reconstructed 169,387 sessions by grouping according to associated session identifiers. We ran word2vec [122] to obtain a vector representation with 100 dimensions for each unique command. We then computed the similarity score between two commands as the cosine similarity between their corresponding vector representations. We conducted a sanity check of the learned command vectors using the Google Embedding Projector [158] and found the results to be plausible, as discussed in Supplemental Section C.I.I.

4.5.2 Layer Similarity

We compute layer similarity using rules that we manually generated based on the combination of observations of participant sessions and additional conversations with PS experts about their use of layers. We classify the strength of the relationship between two active event layers A and B, into five ranked categories and assign heuristic values accordingly, as shown in Table 4.1. Users frequently operate on the same layer consecutively for a while before switching to a different one, and we assign the maximum similarity value of 1 between these identical layers. The user often copies a layer to a new one to make repeating objects, edit repetitively, or back up a layer, so we assign an 0.8 similarity value between a duplicate and its source layer. An adjustment layer applies color and tonal adjustments to one or more main layers without permanently changing pixel values, a common technique for non-destructive image editing. Layers stack up on each other in pixel space, and although technically all layers below another could be affected we determined that
users often consider only directly above/below pairs, so we assign a measure of 0.5 similarity between the two. For layers in the same group, we assign a value ranging from 0 (infinitely distant layers) to 0.5 (direct sibling layers) depending on layer proximity in the hierarchy: layers separated by $d$ hops have similarity $1/2^{d-1}$. Finally, all other layers that differ are assigned with the minimum similarity 0. We summarize this description with detailed formulae in Supplemental Section C.1.2 and provide annotated examples of layer hierarchies in Supplemental.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Description</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same layer</td>
<td>$A = B$</td>
<td>1.0</td>
</tr>
<tr>
<td>Duplicate layer</td>
<td>$A$ is a copy of $B$</td>
<td>0.8</td>
</tr>
<tr>
<td>Adjustment layer</td>
<td>$A$ is an adjustment layer of $B$</td>
<td>0.5</td>
</tr>
<tr>
<td>Grouped layer</td>
<td>$A$ and $B$ are located in the same layer group</td>
<td>$\leq 0.5$</td>
</tr>
<tr>
<td>Other diff. layer</td>
<td>none of the above</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table 4.1:** The five kinds of relationships between two layers $A$ and $B$ and their assigned similarity values. Each relationship is commutative.

### 4.5.3 Image Diff

The image difference (diff) of two events is the percentage of different pixels between the two corresponding images out of the total number of pixels in an image. The implied conjecture in previous work is that in an image editing session the amount of change, as quantified using image diff, is relevant to segmentation where a larger image diff implies greater difference between two events. Although there exist more advanced image diff scores such as the structural similarity index (SSIM) [181], we do not choose them because they are potentially biased to a particular type of image edits: for example, SSIM is sensitive to structural changes but not color tone changes.
4.5.4 Working Region Overlap

We also compute a second potentially relevant image-related feature, to assess which of the two is a better indicator. The working region overlap measures the degree of continuance in pixels between two events. It is defined as the percentage of overlapping changed pixels between two events out of the total number of pixels of an image. We conjecture that two events are similar if they operate on a similar pixel area, and thus that they would be likely to belong to the same chunk. We compute it as a “diff of a diff”: count the different pixels between the diff image of two events down-sampled by a factor of 32, where a diff image of an event is the difference image between itself and the previous one. We down-sample to check for overlap in coarser regions rather than individual pixels.

4.5.5 Duration

The duration is simply the time between consecutive events.

4.6 Segmentation Model

Our two-stage model predicts low-level chunk boundaries with an SVM classifier, and has two alternatives for computing higher-level chunks from those results.

4.6.1 Low-level Segmentation

To classify events as boundary or non-boundary with an SVM, we must construct a feature vector, partition the data, consider the appropriate point in the precision-recall trade-off space, and tune the hyperparameters.
Feature Vector Construction

To determine whether an event is the starting point of a chunk, a reasonable way is to compare it against its previous events in terms of the computed similarity features. If the difference is big enough, it is likely to belong to a new chunk. We consider a window of $k$ previous events for comparison, where the window size $k$ as a hyperparameter that will be tuned. For each previous event in the window, we use the five similarity (or distance) measures, described in the previous section. We then concatenate all measures of comparison to all previous events in the window into a feature vector, whose size is $k \times 5$.

Data Partition

We split our collected dataset (5718 events in 16 sessions) into training, validation, and test sets. By the second round of data collection we had finalized what features and model to use; we set aside 5 of the 8 sessions that second round as the test set, and used the other 11 sessions for training and validation (3 from the second round, and all 8 from the first round). The 5 test set sessions are randomly drawn with stratified sampling for task type: 2 for poster creation, 2 for portrait retouching, and 1 for special effect creation. This procedure results in a total of 1842 events (about 30% of grand total) in the test set. The remaining 3860 events are further split randomly into training (75%) and validation (25%) set.

Precision Recall Trade-off

Our dataset is unbalanced, with many more non-boundary events than boundary events. We consider the trade-off between precision and recall in model training and validation, and deliberately trade precision for higher recall. It is important to correctly identify as many boundaries as possible, and false negatives (boundaries predicted as non-boundaries) are undesirable as it is not easy for end users to identify these missed boundaries. On the other hand, false positives (predicting non-boundaries as boundaries) are less detrimental: although these errors lead to
over-segmentation of the logs, users can quickly dismiss them through an interactive interface.

**Hyperparameter Tuning**

We train the model on the training set, and measure performance on the validation set to tune these hyperparameters, following common practice in machine learning. Supplemental [Section C.3] covers the procedure and results in detail. In brief, the first hyperparameter, window size $k$, is the number of previous events used in the feature vector. Using receiver operating characteristic (ROC) curves, we find that window size has little influence on the model performance, and thus we choose $k = 1$ to reduce the size of feature vectors. The second one, threshold $t$, determines the probability value where an event is predicted to be a boundary. We select $t = 0.24$ in consideration of the trade-off between precision and recall.

### 4.6.2 High-level Segmentation

The higher-level chunks are built based on the predicted lowest-level chunks. The higher-level segmentation should be flexible in granularity, to accommodate different downstream use cases. We describe two methods to construct high-level chunks: multi-tier thresholds in SVM and agglomerative clustering.

**Method 1: Multi-tier Thresholds**

In the SVM model for lowest-level segmentation, the threshold $t$ can be interpreted as the granularity of segmentation: as $t$ increases, the granularity gets coarser and there are fewer predicted boundaries (chunks). In that case, we use a threshold that achieves the optimal performance on a labeled dataset (the validation set). Intuitively, we can simply use a series of higher thresholds to obtain multiple higher levels of chunks despite the lack of ground truth. Every higher-level boundary will necessarily fall along a lower-level boundary, since the same probability scores are
Method 2: Agglomerative Clustering

Higher-level chunks can also be generated from lowest-level ones using the bottom-up approach of agglomerative clustering [192]. This algorithm uses a predefined similarity metric to iteratively find the most similar pair of adjacent chunks and merges them together, until a single remaining chunk contains the whole session. We define the similarity between two chunks as the average similarity of all pairs of events across the chunks. We compute a weighted sum of the five feature values, where the weights are the linear coefficients of the kernel function in the learned SVM model, since the coefficients indicate feature importance. The output of the agglomerative algorithm is a binary tree, where each node represents a chunk and the distance to root can be considered as its granularity. Therefore, we can make a cut through this tree to obtain chunks of the desired granularity, either a straight cut with uniform distance to the root or a more complex shape to capture subtrees of different resolution.

Comparison

Both methods reuse some parts of the low-level SVM model: the multi-tier threshold method reuses the probability score directly, whereas the agglomerative method leverages the linear coefficients in the kernel function.

We compare the two methods in terms of computational complexity, potential limitations, and usage pattern. For computational complexity, the multi-tier threshold method does not need extra computation: it is $O(n)$, where $n$ is the number of events. The agglomerative method has to compute event and chunk similarity iteratively. Its computational complexity is $O(n^2)$, an acceptable cost since a session contains a few hundred events.

A potential limitation of the multi-tier threshold method is that the SVM model
for low-level segmentation only leverages minimal context information (one previous event), which has not been validated for the high-level one. In contrast, the agglomerative clustering computes average values over many event pairs as the chunk similarities, making it more robust against single-event outliers. However, the agglomerative method may yield sub-optimal results due to the greedy merging mechanism [48].

Thresholds in the multi-tier method are straightforward to specify but less flexible as one value is used across all levels of the hierarchy. The cluster hierarchy resulting from the agglomerative clustering method provides the flexibility for an adaptive criterion that can vary within a session to select nodes according to the specific use case needs.

4.7 Results: Feature Relevance

The performance of segmentation model highly depends on the effectiveness of the selected features, and moreover one of our goals is to understand feature relevance for its own sake. We analyze relevance to low-level segmentation quantitatively and qualitatively, and the results are consistent. Then we discuss the reason of their (ir)relevance and limitations.

4.7.1 Quantitative Analysis

The linear coefficients in the SVM model, listed in Table 4.2, are usually interpreted as feature importance. The sign of a coefficient indicates whether correlation between the feature value and probability of being a boundary is positive or negative: only duration has a positive correlation, e.g., longer duration indicates higher probability that the event is at a chunk boundary. The absolute value of the coefficient indicates the degree of importance, and we bin these quantities into three categories: commands as most important, layers and duration as important, and the two image-related features have almost no effect.
Figure 4.2: Distributions of the five feature values of non-boundary (top) and boundary (bottom) events. The X axis represents the value of each feature (bin size = 0.05), and the Y axis the counts. Note that the Y scales are 10x different between top and bottom because there are significantly more non-boundary events. We annotate the differences or similarities between the two distributions for each feature.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Coefficient</th>
<th>Relevance category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command similarity</td>
<td>-2.57</td>
<td>Most important</td>
</tr>
<tr>
<td>Layer similarity</td>
<td>-1.74</td>
<td>Important</td>
</tr>
<tr>
<td>Duration</td>
<td>1.55</td>
<td></td>
</tr>
<tr>
<td>Working region overlap</td>
<td>-0.18</td>
<td>No effect</td>
</tr>
<tr>
<td>Image diff</td>
<td>-0.07</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Corresponding coefficients to features in SVM model.

4.7.2 Qualitative Analysis

We also conduct a qualitative analysis of feature relevance. We visualize the distributions of feature values using histograms and compare the distributions between non-boundary events and boundary events to see if the difference is salient. Figure 4.2 shows these distributions, with annotations including our assessment of when they differ and thus the feature is relevant. We can see clearly that layer similarity, command similarity, and duration are relevant, while image diff and working region overlap are not.
4.7.3 Result Interpretation and Limitations

Below we discuss the potential causes for feature (ir)relevance and the limitations of our feature choices.

Command Similarity

It has been shown in the CommandSpace system [3] that the command usage semantics can be learned from large-scale of command logs, analogous to text data, and that the usage semantics are relevant to user goals and subtasks.

However, we see that except for the maximum similarity when commands are identical, the variance in the remaining values is small; most of the range between 0 and 1 is not exploited. A second problem is that a single command in PS such as BrushTool can support multiple different functions, which is analogous to the polysemy problem in natural language. Future work could experiment with a context-aware learning model instead of word2vec, or augment the computation by considering the command parameters (such as brush size) that are not captured by our current PS instrumentation plugin. The third potential problem is the information loss in the computation of cosine similarity between the command vectors.

Layer Similarity

Our observations during data collection were that participants keep the layer panel visible at almost all times as they refer to layers frequently, and are able to explain the semantics of each layer explicitly. Different subtasks in a task are often performed on different layers, particularly when the artwork consists of objects from different source images.

However, we heuristically assigned discrete values as layer similarity for our 5 chosen cases, and these values have not been strictly verified. Also, our current approach does not fully exploit all information available in the layer hierarchy.
We have not found a good way to use the logged information about layer order, which might have further distinguishing power. Also, some layer attributes are not logged, such as layer mask, blending options, and transparency. Exploration of these issues is an important direction of future work.

**Duration**

We observed during the data collection study that pauses do indicate thinking time when switching to a new subtask and when comparing current results to a reference image. We also saw clear evidence of short duration values indicating continuous operations when they are actively involved with a subtask, especially when they have a mental map of how the subtasks should be fulfilled.

However, we think that the duration from our data collection in a lab setting may be artificially clean, since participants are doing an assigned task under observation. We conjecture that in real-world usage, pauses may often arise from external interruption rather than an internal switch of subtasks. Therefore, we conclude that the relevance of duration is debatable.

**Working Region Overlap**

Our results contradict previous work from Chen et al. [26] that uses working region overlap. We conjecture that the signal from pixels of image content is dominated by the one from layers, which serve as an abstraction of visual object structure, for identifying subtasks. Another possible reason for our opposite finding could be the information loss with pixel counts: we use a single number instead of a vector representation in Chen’s paper.
Image Diff

We conclude that the image diff is too noisy for segmentation. A subtask switch (boundary event) does not always come with large change in pixels; for example, adding text in a small font only changes a small number of pixels. Similarly, large changes in pixels sometimes can happen within a chunk, for example moving a large object. We see from the distributions that most events change very few pixels.

4.8 Results: Low-level Segmentation

With reliable ground-truth labels for the low-level segmentation, we measure model performance on the test set, conduct an preliminary analysis on the variance across different tasks, and compare our performance metrics with a previous paper.

4.8.1 Performance Metrics

We train the SVM model on the training set, tune hyperparameters on the validation set (window size \( k = 1 \), threshold \( t = 0.24 \), and measure performance on the test set, which contains 1842 events. The F2 score, which weighs recall twice as much as precision, is 0.76, with a decent recall 0.87 but modest precision 0.51. The numbers are almost identical to the performance metrics on the validation set (provided in Supplemental Section C.3), suggesting that the model does not overfit.

4.8.2 Task Variance

We analyze the variance of model performance across different types of task (poster creation, portrait retouching, and special effect creation) to verify our claim that our model generalizes.

We categorize all 16 collected sessions by task type, and conduct two analyses to measure model performance. In Analysis 1, we use the chosen threshold \( t = 0.24 \)
for all three tasks, and compute the F2, recall and precision scores for each task. In Analysis 2, we tune the threshold hyperparameter for each task. We observed that different participants organize their sessions at different levels of granularity, which is particularly obvious across different task types, so we need to understand how the threshold affects performance for different task types.

We present the performance metrics in Table 4.3. For both analyses, the difference of F2 scores between any two task types is low (±0.08). Within each task type, the difference of F2 scores across analysis is also low (±0.03). In Analysis 2, where the threshold is optimized for each task type, we notice that the threshold for portrait retouching (0.17) is different than the other two (0.27, 0.25), confirming our observation about the influence of task type on granularity. In conclusion, the SVM model can generalize to the three task types, while further research is needed.
to address the nuances in granularity.

<table>
<thead>
<tr>
<th>Task type</th>
<th>Poster creation</th>
<th>Portrait retouching</th>
<th>Special effect creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data scale</td>
<td>#sessions 8</td>
<td>#events 3156</td>
<td>#events 6</td>
</tr>
<tr>
<td>Analysis 1</td>
<td>F2 recall 0.71</td>
<td>F2 recall 0.77</td>
<td>F2 recall 0.79</td>
</tr>
<tr>
<td></td>
<td>precision 0.85</td>
<td>precision 0.82</td>
<td>precision 0.97</td>
</tr>
<tr>
<td></td>
<td>threshold 0.43</td>
<td>threshold 0.63</td>
<td>threshold 0.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.24</td>
</tr>
<tr>
<td>Analysis 2</td>
<td>F2 recall 0.73</td>
<td>F2 recall 0.80</td>
<td>F2 recall 0.79</td>
</tr>
<tr>
<td></td>
<td>precision 0.82</td>
<td>precision 0.92</td>
<td>precision 0.96</td>
</tr>
<tr>
<td></td>
<td>threshold 0.49</td>
<td>threshold 0.52</td>
<td>threshold 0.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 4.3: The performance metrics of sessions in the three task types.

4.8.3 Comparison to Previous Work

The closest work to our model is Chen et al.’s method [26]. They also trained an SVM model, targeting smart undo as a primary use case. We deemed it infeasible to directly compare with their model fairly. It does not make sense to apply their trained model on our dataset, because their training dataset (and hence their model) only has portrait retouching data, whereas we covered three tasks in total. Also, a difference in data collection (they have data on brushing area but we do not) induces technical difficulties in re-implementing their model for retraining on our dataset.

Chen et al. chose window size $k = 5$ for their model and achieved a similar F2 score (0.74) with a lower recall (about 0.7) but much higher precision (0.96) on their dataset with portrait editing task only. That outcome is surprising, since they did make similar arguments to our own for valuing recall over precision. We also note that our segmentation problem is considerably harder: we deal with three types of tasks instead of portrait retouching only, and we did not prescribe a set of pre-defined subtasks for the participants like they did. We argue that our results are reasonable, since we tackle a harder problem. In conclusion, we have achieved
reasonable results for a more difficult problem than theirs.

4.9 Results: Multi-level Segmentation

We present the visualizations of two exemplar sessions to show how our computed low and high-level segmentation aligns with human labels, which are subjective judgements from the authors. Here we show two from five sessions from the test set: one portrait retouching session (S10), and one poster creation session (S15). They are representative sessions in terms of both general model performance and failure cases; Supplemental Section C.4.2 contains the analysis of the other three test sessions. We also provide high-resolution screenshots for all sessions in supplemental materials. We first provide an overview of participant’s workflow (high-level segmentation in the human labels), and then inspect how our computed results align with human labels.

4.9.1 Exemplar Retouching Session

The goal of the first exemplar session (Figure 4.3(a)), with 180 events, is to restore an old photo (a child’s portrait) with cracks and stains on it. At the bottom of the figure there is a scatterplot showing the probability scores of events being predicted as a boundary in time order, with red dotted lines to show threshold values. Above the scatterplot are three versions of segmentation (computed chunks by the multi-tier thresholds SVM in red, computed chunks by the agglomerative algorithm in blue, and human labels in gray). Each colored rectangle represents a chunk, and a white space (gap) between two rectangles represents a chunk boundary.

From the grey human labels, we can see that the participant is working on different regions in the image roughly one at a time: forehead and right eye, right face and ear, teeth and lips, and finally the background. Note that the regions (face, eye, background, etc) are only meaningful for a human user. However, there are also outliers subtasks that do not follow this order: e.g., trying out a different command Healing Brush to fix the photo. In the last step to fix the left background, they
used a more complicated procedure than previous chunks, resulting in a two-level segmentation.

To compare different versions of segmentation, we can inspect the alignment of gaps (boundaries) vertically. In general, the gaps in human labels and those in the computed results align well. One obvious missing boundary happens around the middle between “fix right face and ear” and “fix teeth and lips” (highlighted with a dotted rectangle), due to the high similarities between these events (same layer and command, duration small, and little changes in image content).

### 4.9.2 Exemplar Poster Session

The goal of the second exemplar session (Figure 4.3[b]), which is longer with 378 events, is to create a poster featuring two football players. From the human labels in the figure, we can see that the chunks consist of variant numbers of events, and the hierarchy of chunks is unbalanced; that is, different high-level chunks have different subtree depths below them. From the high-level human labels and an expanded part in Figure 4.1, we can see that the participant did multiple trials for some subtasks, making some mistakes and corrections.

For the low-level chunks between human labels and computed results, most gaps are aligned, indicating a good match between the two. There are very few missed boundaries (highlighted): one example happens when the participant finished making a layer mask and then started adjusting opacity, where they stayed at the same layer and used commands that are semantically close (Brush Tool vs. Master Opacity Change). There is some but not excessive over-segmentation: the consecutive gaps that separates individual events into chunks. For example, it happens in a human-labeled chunk “tune colors of front player”: he actually performed two scaling operations of the image in between color tuning operations, resulting in low similarity between consecutive commands (Free Transform vs. Camera Raw Filter) and thus over-segmentation in the computed results. We will discuss this particular user behaviour in the next section.
For the high-level chunks, although both computed results aggregate most of the single-event over-segmentation, neither has a good match with the human labels: about half of all high-level boundaries are missed. As we have already noted, we do not consider that data to represent ground truth; the ambiguity about what granularity to code for led to the two human coders had even higher levels of discrepancies with each other in the sessions they both coded, so that data was not used in the development of the high-level model. Nevertheless, we inspect the probability scores of the missing boundaries in the scatterplot, and find that many of them have a high probability score, but it is hard to draw a horizontal straight line that could serve as a threshold to separate most of them out. This finding indicates that there is some value in reusing the results of SVM model in the high-level segmentation, but it needs further research to improve the performance.

4.10 Discussion

We discuss the strengths and weaknesses of our segmentation model, of the dataset we collected compared to previously reported ones, and the nuanced characteristics of real-world user behaviour in image editing revealed by our study.

4.10.1 Strengths and Limitations of Model

Our multi-level segmentation model achieves fast computation, due to its simplicity and the small number of parameters. The low-level SVM model takes less than 1 second to train and validate, and prediction is immediate. The high-level segmentation takes at most a few seconds to handle an 1K-event session with a JavaScript implementation. This speed makes the model suitable for streaming log data, the hallmark of the downstream interactive application use cases.

It is hard to thoroughly evaluate our approach to higher-level segmentation since we do not have appropriate labelled data for it, due to the subjectivity of the granularity assessment. Ground truth labels and a more rigorous evaluation would allow further progress. The complex user behaviours described below do leave consider-
able room for improvement in model performance. Our precision is currently only 0.5; although we did prioritize recall, both measures need to be improved.

4.10.2 Dataset Comparison

We now compare our dataset to two other datasets from previous work in more depth. In the nonlinear revision control paper by Chen et al. [25], most of their sessions are less realistic than ours, where there is little complex user behaviour such as continuous refinement, mistakes and corrections, and experimentation. The scale of their data is also smaller in terms of number of sessions and number of events per session: a few sessions with less than a dozen events, and a few sessions of sketching with hundreds of events.

The adaptive history paper by Chen et al. [26] targets only the portrait retouching task, with predefined subtasks chosen from a list. The restricted variance in subtasks makes their model hard to generalize to other task types, or even retouching that does not follow the predefined subtasks.

A major difference between our dataset and previous ones is that we capture layer information. However, there is still some available data that our plugin architecture does not capture, such as command parameters (e.g., brush size) and layer attributes (e.g., transparency). Future work could investigate whether it contains useful signal to exploit for segmentation.

4.10.3 Real-world User Behaviour

Our data collection study documented that the real-world usage patterns of creativity support tools are usually messy and complex: users rarely perform precise and effective actions from the beginning of a session to the end. We characterize several of the complex user behaviours that we observed.

First, users make mistakes. Common examples of mistakes are misuse of commands, undesired command parameters, or operations on a wrong layer. The user
may or may not correct the mistakes, and corrections may happen immediately after a mistake or later. This situation is challenging as the user usually does not consider these corrective actions as a change of subtask, but the model would only see the a sudden change of events and thus predict them as boundaries. Second, users interleave different subtasks together. This situation may happen when the user has no clue of how to perform a subtask, when the user realizes there is another subtask that is more urgent than the current one, or when the user finds a previously done subtask unsatisfying. This situation is challenging for segmentation as the user’s mental model is less organized. Third, users experiment with the tool to achieve a desired outcome through trial and error. This situation is common in users with lower levels of expertise, and the resulting session usually contains many undo actions and layer deletions. The challenge in automatically segmenting this kind of user behaviour is that the difference between trials can be subtle and hard to detect. Fourth, the boundaries between subtasks could be intrinsically fuzzy. Before switching to a new subtask, users often check if results are satisfactory by toggling layer visibility, or make small adjustments of different components in the artwork while thinking about what to do next. These actions result in fuzzy boundaries between subtasks that span multiple events, but our model assumes single-event boundaries.

Our data collection study succeeded in revealing a picture of user behaviour that is far more complex than what previous work had captured, where less attention was paid to ecological validity. Our model makes a significant step towards addressing this complexity, but there are many interesting problems to address in future work.

4.11 Summary

We developed a multi-level segmentation model for real-world image editing logs that works for three image editing tasks. We present evidence for what features are relevant and irrelevant for the segmentation. Results show that command and layer similarity are highly relevant, image-related features are not useful despite claims in previous work, and duration is debatable due to the limitation in data collection.
We also present quantitative and qualitative evaluation of our segmentation model, and show that it performs reasonably well for the challenging problem of segmenting realistic logs that capture mistakes, experiments, and subtask switching.
Chapter 5

CorGIE: Visualizing Graph Neural Network by Exploring Correspondences between a Graph and Its Embedding

Graph neural networks (GNNs) are a class of powerful machine learning tools that model node relations for making predictions of nodes or links. GNN developers rely on quantitative metrics of the predictions to evaluate a GNN, but similar to many other neural networks, it is difficult for them to understand if the GNN truly learns characteristics of a graph as expected. We propose an approach to corresponding an input graph to its node embedding (also known as latent space), a common component of GNNs that is later used for prediction. We abstract the data and tasks, and develop an interactive multi-view interface called CorGIE to instantiate the abstraction. As the key function in CorGIE, we propose the K-hop graph

This chapter is a slightly modified version of the following publication: Z. Liu, Y. Wang, J. Bernard, and T. Munzner. Visualizing Graph Neural Network with CorGIE: Corresponding a Graph to Its Embedding. IEEE Transactions on Visualization and Computer Graphics (TVCG). Under review [113].
layout to show topological neighbors in hops and their clustering structure. To evaluate the functionality and usability of CorGIE, we present how to use CorGIE in two usage scenarios, and conduct a case study with two GNN experts.

5.1 Motivation

Graph neural networks (GNNs) are machine learning (ML) models that have received substantial recent attention due to their ability to deal with abstract concepts like relationships and interactions. GNN models are widely used in downstream ML applications such as fraud detection, traffic modeling, and product recommendation, in addition to the classic ML application domains of natural language processing and computer vision.

During training, a GNN model takes in a node-link graph and as output generates a **node embedding**; that is, a representation of all discrete nodes in the graph as fixed-dimensional vectors in a continuous **latent space**. Proximities between embedded nodes in the latent space represent meaningful similarities between nodes in the input graph learned during the training. The node embedding leverages both the topology – connectivity between neighboring nodes – and node features – information associated with each node – of the input graph. (Information associated with nodes, typically called node features in the ML literature, is sometimes called node attributes in the visualization literature.) The node embedding is then available for feeding into downstream ML applications, for example to make predictions about nodes and links. [Figure 5.1a summarizes the standard GNN pipeline.](#)

To achieve optimal performance, model developers must evaluate whether the training has succeeded in producing a GNN model that has in fact learned the important characteristics from the input graph. Model developers often need to determine the stopping criteria for training a GNN model; in other words, to answer the question “are we there yet?” when training models or tuning hyperparameters. In addition to global questions such as when to stop training and tuning, model developers need to debug and trace errors and sub-optimal states in the trained model. For example, they may want to identify and understand the groups of nodes that suffer
Figure 5.1: CorGIE motivation. (a) Standard pipeline for graph neural network (GNN) training and usage. (b) Previous approaches to evaluate GNNs mostly focus on the predictions [40, 167, 188], or to separately inspect the node embedding; (c) our approach with CorGIE is to directly support exploration of correspondences between an input graph and its node embedding.

from mis-classification the most. In some cases, they also need to compare results from different model architectures or hyperparameters to choose the best ones for production use.

The most prevalent current approach for model evaluation is to compute quantitative metrics based on the downstream ML applications, such as precision and F1 for node classification and Hit@k for link prediction, which are used for cross-validation during training [40]. Some researchers have proposed algorithms to compute explanations for prediction results in terms of influential nodes, or what features are important in the determination of a specific node [188]. Another common approach is to conduct qualitative sanity checks, such as manual inspection of node or link instances, for example by visualizing with multi-class scatterplots of the dimensionally reduced node embedding [167]. Figure 5.1b summarizes these previous approaches to evaluation.

However, these existing evaluation approaches fall short because they do not sufficiently support developers in directly understanding the correspondences between input graphs and their node embeddings. Developers must contend with a GNN model as a black box, because it is hard to check whether all important information is codified in the embedding. Even if they find errors in the downstream predicted node labels, it is hard to even trace these errors back to the upstream GNNs, or to refine the GNNs to avoid them.
Our key idea is to aid GNN evaluation by surfacing the correspondences, or lack thereof, between an input graph and its node embedding produced by a GNN. Figure 5.1c illustrates this approach. Geometrically speaking, a GNN should learn to place graph nodes with similar neighbors and feature values in a nearby location in the high dimensional latent space. Therefore, we can evaluate a GNN by verifying how well it preserves the characteristics of the input graph in the node embedding. For instance, one can explore how a cluster in the node embedding corresponds to the similarity of topology and/or node features in the input graph. Conversely, one can explore how close two nodes sharing many topological neighbors are represented in the node embedding. To achieve these goals, we present three contributions.

Our first contribution, presented in Section 5.3.3, is a data and task abstraction for visually exploring correspondences between an input graph and the derived node embedding, to understand if a GNN model has learned important characteristics of the input graph to construct the embedding. The abstraction is the result of extensive iterative refinement.

To instantiate the abstraction, our second contribution is the design and implementation of an interactive multi-view interface, CorGIE (Corresponding a Graph to Its Embedding). CorGIE is agnostic to specific GNN models, with a “grey-box” approach that ignores the internal model details like neurons and weights and only assumes that the GNN aggregates information from neighboring nodes. We describe the design of CorGIE in Section 5.4.

To fulfill the important task of showing topological neighbors for nodes of interest, our third contribution is a new visualization technique, the K-hop graph layout. The K-hop layout reveals node neighbors hop by hop, similarly to how a GNN aggregates information, and reveals the clustering structures within the node neighborhoods. Previous graph representation and layout algorithms do not suffice for this scenario. We present the details of this technique in Section 5.5.

We validate our claims in two ways. We provide two usage scenarios that walk through how the CorGIE design supports the task and data abstractions in detail.
We also discuss the results of a study with two GNN experts that provide preliminary evidence of the utility and usability of CorGIE.

5.2 Background

We first provide the minimum necessary technical background on GNNs to make this work self-contained. We then describe two concrete usage scenarios that motivate CorGIE. We defer our discussion of the related work to Section 5.7.

5.2.1 Graph neural networks

The widespread encoder-decoder perspective is a useful starting point for understanding general graph neural networks [23]. The encoder combines topology, and optionally node features, to produce a node embedding. The decoder takes the node embedding to make inferences in downstream ML applications. In the GNN literature, the node embedding is an intermediate representation within the pipeline that is not necessarily of direct interest outside it. Our perspective is different: we emphasize inspection of the learned embedding as the key to understanding. Moreover, the downstream ML applications are usually considered as an intrinsic component of the GNN pipeline. In contrast, we separate the downstream ML applications from the preceding pipeline so that CorGIE can support multiple downstream applications. CorGIE’s agnostic approach supports both node classification that predicts properties of single nodes, as used for movie tagging or document topic labeling), and link prediction that predicts properties of node pairs, as used for movie recommendation.

A GNN is usually trained using either node labels or node connections depending on the application. For node labels, training can be supervised by separating the nodes into training, validation, and test sets if all node labels are available, or semi-supervised if only a small number of labels are available. For node connections, training is unsupervised. Two nodes connected by an edge are considered a positive node pair, whereas a negative node pair has no edge between them; both positive
and negative pairs are sampled during training.

A GNN learns from the input graph with neighborhood aggregation (also known as message passing), where the embedding of a node is computed by traversing to its topological neighbors to collect and aggregate node features from them. A GNN has multiple layers, allowing aggregation to take place hop by hop: each node aggregates the output of its neighbors from one layer to the next. The input layer takes the node features, the last layer outputs a high dimensional node embedding, and the $k$-th layer aggregates from the $k$-th hop neighbor. The number of layers $K$ is usually small, with $K \leq 3$, indicating that it is a shallow network. GNNs differ in the aggregation functions, ranging from a simple mean to complicated non-linear functions [133]. Our approach takes $K$ as a parameter, but is fully agnostic to the details of the aggregation function.

### 5.2.2 Motivating scenarios

In this chapter, we refer to the target users of CorGIE as GNN developers, ranging from users who consume pre-trained GNNs to experienced researchers who can propose new model architectures.

Before diving into the details of our approach and tool, we present two representative scenarios of a hypothetical GNN developer, Alice. For each, we introduce a dataset, which consists of an input graph and the embedding constructed by the trained GNN model, and a set of visualization tasks that apply to it. We provide extensive additional examples of concrete tasks in Supplemental Section D.1.

**Movie: recommendation**

Alice has a bipartite movie-user graph\(^2\), where an edge indicates that a user has watched or rated a movie. Node features in this graph depend on the node type: the movie node features are budget, popularity, vote average, #cast, #crew; the

\(^2\)Extracted from a Kaggle dataset: [https://www.kaggle.com/rounakbanik/the-movies-dataset](https://www.kaggle.com/rounakbanik/the-movies-dataset)
user node features are vote average and #votes. The goal is to recommend movies to users; that is, the downstream ML application is link prediction. Alice trains a GCN (graph convolutional network, a well-known GNN model [93]) with the node connections and node features in the bipartite graph, to produce a 16-dimensional node embedding. She chooses the hyperparameter setting of 16 based on her previous experience.

To evaluate the training results, Alice wants to understand what the GNN has learned. Besides the typical quantitative metrics like accuracy, she wants to examine:

1. The overall clustering structure of the node embedding, to answer questions like *Are the movies separated from the users? Are there clusters of users?*

2. Within a user cluster, she is interested in why nodes are grouped together: *Do they watch similar movies; that is, do the movies share first hop neighbors? Do they rate movies with similar scores? Do movies they have watched have similar budgets?* Similarity in topology and/or feature values indicate the GNN has done a good job in grouping these users.

3. Between two user clusters, she wants to see their differences: *Do they watch different sets of movies? Do they rate movies differently?* Significant differences indicate that the GNN has done a good job in differentiating the two user groups.

4. Instance-level inspection of movie recommendations is Alice’s next undertaking. She wants to spot-check the list of recommended movies for some user, for example a recommendation of *Interstellar* for a viewer who has seen *The Matrix* and *Guardians of the Galaxy*. She wants to understand why the GNN has generated them: *Are recommendations determined by shared viewing patterns, as with collaborative filtering [149]? Are they motivated by similar node features? Do they capture topic similarity, like science fiction?*
Cora: paper topic labeling

The Cora dataset is commonly used as a benchmark in the GNN literature [116]. It is an academic paper citation graph, where the node features are binary vectors indicating what words from a 1433 word dictionary exist in the paper represented by a node. The downstream ML application is to classify the papers according to a set of given topics (classes). Alice trains a GAT (graph attention network, another well-known GNN model [175]) with the node labels in the Cora graph, and produces a 7-dimension node embedding because there are 7 classes of paper topics. She thus converts the values for each dimension to probabilities of belonging to that class, which is a common practice in the GNN literature.

Following a similar analysis process for the movie scenario, Alice first explores the clustering structure by visualizing shared topological neighbors and similarity of node features from an overview level. Then, she tries to understand the pattern of misclassified nodes. She selects groups of these nodes that are either located in the same area of latent space or have the same predicted label; that is, the same error. She inspects the shared neighbors and the words within groups to determine whether there are commonalities to blame for the misclassification.

5.3 Data and task abstraction

After considering many examples of concrete tasks faced by GNN developers, we generalized them to understand the problem at a higher level. We formed the data and task abstractions iteratively, over several months, through four parallel thrusts: review of the GNN and visual analytics literatures, informal interviews of four GNN developers within three different organizations, development and use of prototypes of the CorGIE interface, and our reflection.
5.3.1 Overview

We generalize the many concerns faced by GNN developers as they train their models into three driving questions. For memorability, we frame them as questions that might be asked on a road trip:

- Q1: are we there yet: should we train or tune more?
- Q2: are we lost: does it behave as we expect?
- Q3: what’s that: what does this exemplar do?

The first big-picture question concerns the potential need to keep training the model for more epochs, tuning the hyperparameters, or reconsider how they construct the graph. It would be answered by understanding whether the results are satisfying. The second question, also big-picture, concerns developers’ understanding and trust in whether the GNN learns inherent characteristics from the input graph; that is, does it behave as they expect. These two big-picture questions are particularly difficult to answer with previous evaluation approaches, which rely heavily on quantitative metrics for the downstream ML application; we aim to provide visual and qualitative evidence and insights to answer these two questions in a more detailed and thorough way. The third question is more specific, concerning investigating exemplar instances of nodes or links. Some previous approaches like the GNNExplainer [188] do support such single-instance inspections, but we seek to make them easier and faster.

We posit that finding the correspondences among 1) the node positions and clustering structure in the latent space, 2) the topological neighbors in the input graph, and 3) the node feature distributions in the input graph, can shed light on these questions. We thus discuss three abstract data spaces, as illustrated in Figure 5.2: latent space, topology space, and feature space.
Figure 5.2: Data and task abstraction. Users can specify items within all three data spaces, and investigate correspondences between the latent space and topology or feature spaces. The targets of activity in each space are listed below it.

5.3.2 Data Space Abstraction

Our data abstraction features three sub-spaces: the latent space, the topology space, and the feature space. We compute distance metrics in each of these spaces. We also incorporate prediction results if available. We follow the order of views in the CorGIE interface to introduce the three spaces.

Latent space

The latent space is where the node embedding learned by the GNN lives. Each node in the input graph has its own high-dimensional vector representation. The number of dimensions is usually less than a thousand, although this number does not affect the complexity of the tasks and thus is not a constraint for the CorGIE design. CorGIE assumes $K \leq 3$, where $K$ is the number of layers in the network.

In the latent space, the absolute vector values are usually not directly interpretable. We are interested in the vector similarities and particularly, whether there is a clustering structure among the node vectors. For example, in the Cora scenario, a relevant question is whether there are exactly seven clusters of papers in the latent space, which would match the seven topics in the ground truth.

Some GNNs can generate edge embeddings or graph embeddings, but the scope of this work is node embedding and others are left for future work.
Topology space

The topology space consists of the topological connectivity of one single input graph. The input graph could be either homogeneous, with a single node type, or heterogeneous, with multiple node types. For example, there are two node types in the Movie scenario: user and movie. The design target for CorGIE is to handle up to 6 node types, which covers many useful cases for heterogeneous graphs [185]. For graph size, CorGIE will accommodate up to 20K nodes, to handle many prevalent benchmark datasets in the GNN literature [40].

In the topology space, we are interested in the topological neighborhoods and the connections (edges) between node pairs. GNNs aggregate information along edges to compute the node embedding. We thus derive each node’s neighbor set containing its k-hop neighbors; that is, the set of all of the nodes reachable from a node $v$ within $k$ topological hops.

We limit our scope to one input graph, leaving the multiple graph case to future work. We also leave the support of multiple edge types for the future.

Feature space

The feature space consists of all features (often referred to as attributes in the visualization literature) of the graph nodes. We distinguish between dense and sparse features. Dense features are numeric or boolean features that can be collected and understood independently, with interpretable semantic meanings for each one, such as budget or average vote from the Movie scenario.

Sparse features are usually collected together and share a combined semantics, where interpretation typically takes place across the entire set considered as a whole. Typically they are stored as one-hot encoded categorical features like city names or the dictionary of words from the Cora scenario, where the collection is represented by hundreds or thousands of bit vectors.

In the feature space, we are interested in the distributions of feature values, which
involves deriving aggregations. We deep-dive into the details of feature aggregation in Section 5.4.2.

The design target of CorGIE is to handle up to a dozen dense features and up to 2K sparse features. Note that we deem edge or graph features out of the scope of this work.

Distance metrics

To quantify correspondences between spaces, we introduce a distance metric between any pair of nodes within each space, so that scalar distance values across different spaces can be compared. In the latent space, we derive the cosine distance of the embedding vectors, which is commonly used in the downstream ML applications. In the topology space, we derive a topologically oriented distance measure, the inverse Jaccard index for the full set of k-hop neighbors of each pair of nodes in the input graph: \[ 1 - \frac{|A \cap B|}{|A \cup B|} \] where \( A \) and \( B \) are the \( \leq k \)-hop neighbors of the two nodes. It is intuitive and easy to compute, but the flattened neighbor sets are sometimes an oversimplification because neighbors at different hops cannot be distinguished from each other. In the feature space, we derive the Euclidean distance of feature vectors (scaled linearly to \([0, 1]\)), which is also familiar to GNN developers.

Prediction results (optional)

After training is complete, the GNN can be used for prediction in the downstream ML applications (Figure 5.1). We leverage prediction data by comparing the predicted results to ground truth when available, to derive true/false negatives and positives. For node classification applications, we compare the true and predicted node labels to derive a label correctness value (correct vs wrong) for each node. For link prediction applications, we obtain the predicted positive and negative node pairs, and compare these to the edges of the input graph. We derive labels for two interesting categories of node pairs: false positives, where unconnected node pairs
are predicted as connected (recommended pairs); and false negatives which are node pairs connected with an edge but predicted as disconnected. We also provide options to derive the true positives and negatives if desired. Although the fundamental data and task abstractions do not rely on these prediction results, they can be used for filtering.

5.3.3 Task abstraction

We identify a unifying task abstraction built around an iterative cycle of two phases, as shown in Figure 5.2: specify items in any of the three data spaces, and correspond items between either the latent space and topology space, or the latent space and feature space. Notably, our task abstraction does not entail finding correspondences between the topology and feature spaces, because that exploration would focus only on characteristics within the input graph and would not provide direct insight into GNN behaviors.

The specify step allows users to indicate nodes of interest in any of the spaces, based on the targets visible within each of them. In the latent space, they could specify nodes based on relative positions with respect to any visible cluster structure, or with respect to the latent distance distribution. In the topology space, they could specify nodes based on neighbors, cliques, or the topological distance distribution. In the feature space, they could specify nodes of specific feature values. If available, they can also specify a collection of nodes based on the prediction results such as label correctness or newly predicted links between nodes. Below, we refer to a group of specified nodes as a focal group, which has a double meaning: it is the mental focus within a user’s thoughts at the current step of their data analysis process, and it is the focus of actions and computation within the CorGIE interface.

The correspond step then allows users to explore to what extent the characteristics of the specified nodes in one space correspond to those in the other spaces. For example, for a tight cluster in the latent space, users could verify whether the nodes in the cluster share many neighbors in the topology space and have similar features in the feature space. If so, the GNN has successfully learned how to group these
nodes together. The extent to which these spaces line up with each other is a qualitative judgement call, not a precisely computable metric. The intent of this abstraction is to support GNN developers in obtaining higher confidence and trust in their models than would be possible from purely quantitative summary metrics such as accuracy.

The abstraction supports the process of specify and correspond as an iterative loop, where exploring correspondences can trigger the user’s interest in specifying other sets of nodes. For instance, the user might start by specifying a cluster in the latent space, and after checking its correspondences to the topology space would become curious to then specify a sub-cluster within it to explore further connections. Multiple cycles of refining and changing the sets of specified nodes based on the visual correspondences allow users to connect the dots and find answers all the three of the driving questions.

This task abstraction encompasses all three of our driving questions. It was developed through considering the commonalities between all of the concrete tasks that we analyzed, not only those presented in the motivating scenarios (Section 5.2.2) but also many additional concrete examples summarized in Supplemental [Section D.1] During the abstraction process, we initially considered a much more complex set of targets and actions, such as analyzing outliers with respect to a cluster, or analyzing one node vs. all other nodes, or whether verifying differences should be treated differently than verifying similarities. In the end, we realized that almost all of these questions could be framed much more simply, in terms of specifying a very small number of groups as a target and then checking on correspondences. It was rare to require even three groups: one or two groups were sufficient for almost all analysis questions. Our final design is optimized for up to two groups, although it is possible to specify several more to handle edge cases.

5.4 CorGIE design

Based on the data space and task abstraction, we design CorGIE, a multi-view tool that reveals the correspondence between an input graph and its embedding. We
Figure 5.3: Full screenshot of CorGIE interface on the *Movie* dataset, with two focal groups of user nodes. The views, with names shown in grey boxes, are laid out in four major areas on the screen, shown in blue: (a) the **LATENT SPACE VIEW** is for 2D node positions in latent space; (c) the **NODE FEATURES VIEW** is for feature distribution of all and focal nodes; (f & g) the **TOPOLOGY VIEWS** show local topological neighbors and global topology; (b) the **LATENT NEIGHBOR BLOCKS VIEW** and the (e) **DISTANCE COMPARISON VIEW** connect different spaces. The fifth area is for toggles and menus: (d) the **SETTINGS VIEW**.

describe the design of each view, then the view coordination between them, and
finally the implementation.

5.4.1 Overview

Figure 5.3 shows an overview of the CorGIE interface on the movie scenario, which has seven main views. The **LATENT SPACE VIEW** is on the top left. Two views at the bottom right show the topology space, the **GLOBAL TOPOLOGY VIEW** and a novel graph layout for a user-chosen subset of nodes in the **K-HOP TOPOLOGY VIEW**. The feature space is showcased at the top right in **NODE FEATURES VIEW**. To enable direct and quick correspondence exploration, we have two views in the middle that can incorporate two or three spaces simultaneously. We introduce the **LATENT NEIGHBOR BLOCKS VIEW** to connect the latent and topology space, and
the DISTANCE COMPARISON VIEW to connect all three spaces. Finally, there is a SETTINGS view at the bottom left.

To support the two steps of **specify** and **correspond** in the task abstraction, the interaction design of CorGIE involves three actions: **hover**, **select**, and **focus**.

Node specification occurs through a **focus** action, and correspondence exploration is triggered through **hover** and **select** actions. The **focus** action creates one or a few special node groups of interest, denoted by foc-0, foc-1, and so on. When hovered or selected, nodes are then highlighted across views. When nodes are focused, there will be a novel graph layout showing their k-hop topological neighbors.

### 5.4.2 View design

We introduce the visual encoding and design rationale for each view.

**Latent space**

The **LATENT SPACE VIEW** on the upper left of the interface (Figure 5.3a, Figure 5.6a, Figure 5.8a) shows the clustering structure and the relative positions of nodes in the latent space. We use UMAP [119] to project the latent space to two dimensions and plot the dimensionally reduced nodes as a scatterplot. GNN developers are typically familiar with dimensionality reduction and are thus aware that loss of information is inevitable in this process.

CorGIE supports color coding nodes by their UMAP positions in all views, and in that mode a rainbow background for the entire 2D latent space view provides a salient reference for positions within the latent space, as shown in Figure 5.3. We chose a highly saturated CIELAB colormap (x axis for the variable $a$, and y axis for $b$) that has substantial color exploitation and strong performance for the synoptic localization task of detecting groups of nodes [15]. The combination of dimensionality reduction and this colormap shows latent space distances, so that similar nodes receive similar colors whereas dissimilar nodes receive dissimilar...
Nodes are shape-coded for heterogenous graphs with multiple node types, with the same shape preserved across the latent and topology space views. Edges are off by default to avoid clutter but can be shown on demand in this view, as straight or bundled edges.

Global topology

We present graph topology with node-link diagrams in CorGIE. We adopt the D3 force-directed layout [17] for the GLOBAL TOPOLOGY VIEW (Figure 5.3g, Figure 5.8b). It visualizes the topology of an entire input graph, and is computed once in the pre-processing step. As in the LATENT SPACE VIEW, we encode graph nodes as circles for homogenous graphs and glyphs of different shapes for different node types for heterogeneous graphs. We use straight lines for graph edges and curved lines when edge bundling is enabled.

Although this straightforward backstop layout may provide useful insights in smaller graphs, such global views can be extremely cluttered for larger ones. In that case, our custom layout for exploring local neighborhoods of user-specified subsets of the graph is crucial, in the k-hop topology view.

K-hop topology

The K-HOP TOPOLOGY VIEW (Figure 5.3f, Figure 5.7b & e, Figure 5.8d, Figure 5.9b, Figure 5.10d & e) shows the topological neighbors of the focal node sets specified by users. We devise the k-hop layout algorithm to mimic how information is aggregated in GNNs, with boxes enclosing meaningful groups. On the far left are the focal nodes, with hop-1 neighbors in the middle, hop-2 neighbors to the right, and hop-3 neighbors on the far right. Within the boxes, nodes are clustered using the topological (Jaccard) distance. Full algorithmic details on the layout are provided in Section 5.5.
**Node features**

The **NODE FEATURES VIEW** on the top right (Figure 5.3c) shows feature distributions for up to three sets of nodes: all nodes, and each of the two possible sets of focal nodes (foc-0 and foc-1). Dense and sparse features are visualized differently due to their scale differences.

For dense features, we choose histograms as they are intuitive and well-known for visualizing and comparing distributions of scalar values. They also scale well for the available display space, to handle the design target of up to a dozen dense attributes. We organize these feature histograms as a matrix. Each column represents one feature, whereas the rows show different collections of nodes. The first row shows the full distribution of all nodes; the next row shows the distribution of the focal node set foc-0 and underneath that is the set foc-1. For example, in Figure 5.3c, the two focal groups of user nodes have different distribution in average vote (the second last avr column), but little difference in the number of votes (the last column). Note that for heterogeneous graphs, different types of nodes can have different features, so focal node sets could end up with some empty histograms.

In contrast to dense features, there could be thousands of sparse features, and their ordering is not explicitly meaningful. We designed a two-level custom view to make effective use of the screen space in a compact area, as shown in Figure 5.8f. The top part is a **feature strip** overview that aggregates the information in the more detailed **feature matrix** part below it. The lower **feature matrix** is a heatmap containing one square cell per feature (e.g., a word train), and its luminance encodes the count of nodes with that feature (e.g., the number of papers with the word train in the Cora dataset). Conceptually, the heatmap contains a single very long line that is simply wrapped to fit into a rectangular aspect ratio. It is not a true 2D matrix, so the absolute position of a cell in terms of a row/column is not meaningful. The upper **feature strip** aggregates the information in the heatmap into a highly compressed pixel-based depiction where many consecutive features are combined into the same vertical line, with a luminance representing the maximum values across the range. The number of features to aggregate is determined
by the available horizontal pixel budget of the view, so that the strip fits within it.

As with the dense features, there is a row for all nodes on top, with a row for each of the two focal groups below to show partial feature distributions. In addition, there is a lower \textit{diff} row derived by subtracting the feature values of the two focal node groups. Figure 5.8g shows an example (bottom row) that visualizes differences in the word distribution of \textit{foc-0} and \textit{foc-1}, where the dark strips indicate that there are large differences in word counts.

In each row, the \textit{feature matrix} can be hidden to save space, or expanded to inspect detail. The luminance values are separately normalized within each row since the value ranges could be quite different, so they each require a separate legend.

\textbf{Latent neighbor blocks}

The \textit{Latent Neighbor Blocks View} (Figure 5.3b, Figure 5.8c) overlays topological neighbor distributions on the 2D latent space. The challenge is to show the positions of each node’s neighbors in the latent space, for each of the nodes. We do so with aggregation and nesting. We aggregate by partitioning the space into an $8 \times 8$ grid, to create a coarser representation of the space with 64 \textit{blocks}, and map each node as belonging to the block that encloses it. Within each block, we nest a complete copy of the latent space itself, again at a coarse resolution as an $8 \times 8$ grid of \textit{cells}. Blocks that do not contain any nodes are omitted from the drawing.

This view is inspired by origin-destination (OD) maps for geo-spatial networks [184], but we show neighbor set distributions rather than geographical movement. Within each high-level block, we outline the single low-level cell that corresponds to its block index with red, like the \textit{origin} in an OD map. For every other cell in the block, we encode with luminance the number of neighbors of nodes that fall within the red-outlined origin cell, like the \textit{destinations} in an OD map. If the red-outlined block and its surrounding cells are darker than others in all blocks, that pattern indicates neighbors in the topology space are still neighbors in the latent space, and the GNN has successfully preserved neighborhood structure. Figure 5.8c for the
Cora scenario illustrates this case.

However, there is a limitation to this visual encoding: it is less informative for a bipartite graph, such as the Movie scenario, where movie nodes are only allowed to have user nodes as their first hop neighbors. In this case, as shown in Figure 5.3, the pattern of neighbors separated in the latent space does not reflect evidence of poor GNN performance.

Distance comparison

The DISTANCE COMPARISON VIEW (Figure 5.3e, Figure 5.7e & f) shows distributions of distances in each space and supports comparison between spaces. In the data abstraction (Section 5.3.2), we derive one distance metric in each space. This view reveals matches and mis-matches of distances between the topology and the latent space in one tab, and between the feature and the latent space in a second tab. A match constitutes a positive correlation of the distances in two spaces. This view shows pairs of nodes, which may either represent an edge or be disconnected in the input graph.

To present and compare two scalar distributions simultaneously, we combine two histograms and a gridded scatterplot into one chart, where the x-axis represents the distance distribution in the latent space, and the y-axis represents that in the topology or feature space. The number of items to plot in the scatterplots can be huge, so we use a grid-binning approach to avoid over-drawing, and to speed up computation we also down-sample the node pairs if there are more than a million.

Each tab accommodates multiple charts to show different sets of node pairs side by side, such as all, within a focal group, between two focal groups, or a user-customized filtered set based on connectivity and link prediction values (when available). The customization options are shown in Supplemental Figure D.17, including a choice between linear and log scale to handle the variance in data.

For example, the three bottom histograms in Figure 5.6e show the latent distance
distribution within $foc-0$, within $foc-1$, and between the two groups respectively. We observe that the between-group distances are much larger than the within-group distances, indicating that the two focal groups are two distant clusters in the latent space. Figure 5.9a shows a pursuit of problematic node pairs, where brushing and highlighting the bottom-right area in the scatterplot signals a mis-match with a negative correlation between latent and topology distance.

### 5.4.3 View coordination

To visually underscore the linkages between the views, we carefully maintain the consistency of visual encodings for nodes across views including the shape, color, and size. The node color is user configurable in the SETTINGS view (Figure 5.3d), to either the latent space position (the similarity rainbow), a specific node feature (a sequential ramp), node type (distinguishable categorical colors), or node classification labels if available (also categorical). The shape depends on the node type for heterogeneous graphs, or is a circle in the homogeneous case. The size is also globally configurable in SETTINGS.

We also develop user interactions with consistent semantics across all views. In CorGIE, there are three major interactions from light weight to heavy weight: hover, select, and focus. Typically, they are used in order: hover is for quick and temporary exploration, then select offers a more persistent visual prompt for nodes of interest that are identified with hover, and finally focus stores the currently selected nodes into a persistent group. The look and feel of the interaction is shown in the supplemental video.

The hover action is triggered when users mouse over objects, such as a node or edge in the TOPOLOGY VIEWS, and a block in the LATENT NEIGHBOR BLOCKS VIEW. On hover, CorGIE reacts with strong visual prompts (Figure 5.7c & e): strokes on activated nodes and edges while desaturating the background with a half-transparent mask, black partial distributions on top of the node feature histograms, tooltips and node labels.
The *select* action is triggered by clicking a node/edge or brushing multiple nodes. The visual prompt is highly similar to that for *hover* (Figure 5.6c & d), but persists even when users move the cursor away. Users can control whether the neighbors of target nodes are highlighted during the *hover* and *select* actions, with drop-down menus in the **Settings View**.

The heavy-weight *focus* action enables users to find correspondences for one or two groups of specified nodes. When a focal group is created or modified many views are updated, with changes to the **Features** and **Distance** views and the computationally intensive operation of creating a new **K-Hop Topology** layout. CorGIE supports several focus actions: create a new focal group, add-to/single-out/remove-from existing groups, and clear groups. These actions enable users refine and change the specified nodes, as required by the iterative nature of the tasks (Section 5.3.3). Since only selected nodes can be the target of focus actions, the focus menu to choose one of these actions appears only after nodes are selected. It drops down from the top of the window and looks like a corgi dog’s paw, in keeping with the system name, as shown in the video and Supplemental Figure D.6. The focal nodes are bounded by boxes and labelled with *foc-0* or *foc-1* in both the **Focal Layout** and **Latent Space** views.

### 5.4.4 Implementation

CorGIE is implemented using ReactJS[^3] and Redux[^4] as the frontend scaffold. We choose Canvas over SVG in most of the views for browser performance. For a graph with more than a thousand nodes, there would be thousands of DOM elements such that re-rendering SVGs on user interactions would be very expensive. We develop a layering system using the Konva library[^5] to add visual elements on top of a static canvas in order to avoid the expensive re-render whenever possible. As the *focus* action involves heavy computation that takes seconds and even minutes, we offload it to multiple Web Workers to keep the application responsive, and

[^3]: https://reactjs.org/
[^4]: https://redux.js.org/
[^5]: https://konvajs.org/
use the Comlink library\(^6\) to communicate between threads. We open-source the code at \url{https://github.com/zipengliu/CorGIE}.

### 5.4.5 Design alternatives

During the iterative refinement process for the CorGIE interface, we experimented extensively with different alternatives for exploring the neighbor sets of a single focus group, including an Upset-based [104] view for neighbor sets and a partial adjacency matrix with roll-up histograms. We discuss these design alternatives in detail in Supplemental Section D.3. However, the pixel space required would be infeasible for realistic sizes of graphs (e.g., graphs with hundreds of nodes), and there was no obvious extension to comparing neighbor sets between two focus groups. Our final design relies more extensively on interactive exploration to explore neighbor sets for one or two groups at a time, rather than attempting to visually encode all of possibilities simultaneously. In the case study sessions, a domain expert suggested that we could use the traditional bipartite layout for the Movie scenario, but we consider the force-directed layout a more general solution for all kinds of graphs.

### 5.5 K-hop layout

We present a new visual encoding technique, the K-hop graph layout, which is used in CorGIE’s K-hop topology view (Figure 5.3f, Figure 5.7, & e, Figure 5.8, Figure 5.9, Figure 5.10, & e). We present the computational scalability of the K-hop layout in this section, and evaluate its suitability for the tasks in Section 5.6. We also discuss three design alternatives that we considered and their limitations.

\(^6\)\url{https://github.com/GoogleChromeLabs/comlink}
5.5.1 Algorithm

We first provide an overview of the layout algorithm, then the technical detail and rationale of each step, and finally we discuss its scalability.

The K-hop layout aims to organize topological neighbors of user-specified nodes by the number of hops and their clustering structure. As illustrated in [Figure 5.4], it is computed in four main steps: 1) divide relevant nodes into groups, 2) bound nodes within boxes and lay out the boxes, 3) lay out nodes within each box independently, and 4) perform transformations to optimize global readability. An optional final step is 5) edge bundling.

Divide neighbors. We divide the nodes into groups: user-specified focal groups, all 1st-hop neighbors of focal nodes, all 2nd-hop neighbors of focal nodes, and so on until the Kth hop (where $K \leq 3$). The rest of nodes are deemed irrelevant and thus discarded, so this view shows only a subset of the nodes in the global layout. This division naturally matches the neighborhood aggregation in GNNs (Section 5.2.1), by design. Note that a node can only appear in one group to avoid confusing duplication, with priority given to the leftmost view in which it appears.

Place boxes. We bound the nodes of groups within boxes and place the boxes from left to right. For the focal groups on the left, multiple focal nodes are placed from top to bottom. This layout mimics how the information flow is typically thought about by GNN developers, as we verified in our interviews with them.

Lay out within group. We lay out the nodes within each group independently. To reveal the clustering structure of nodes, we use a dimensionality reduction technique UMAP [119] to reduce a high-dimensional topological distance matrix of
nodes down to two dimensions. Users can choose between local distance (connections to previous hop only) or global distance (all $K$ hop connections). The layout falls back to D3 force-directed layout if there are not enough nodes to run UMAP. Because of the independence between groups, we can parallelize the UMAP processes for performance improvement. It would be possible to use alternative layout algorithms in this step like t-SNE [172]; we choose UMAP for its speed and strength in revealing global structure [29].

**Adjust group.** We need to fine-tune the layout from previous step since purely local layout decisions could endanger the global readability. For example, there can be many edge-edge crossings in step 3 of Figure 5.4, even though the local layout within each box has been optimized. We use an approach similar to Procrustes analysis [16] to potentially reorient each group. Specifically, we can apply six possible transformations to each group: rotation (0, 90, 180, 270 degrees), and flip (horizontal and vertical). These rigid transformations do not change the relative positions within the group. We find the optimal transformations for each group with a simple enumeration algorithm, that is, to enumerate all 6 possible transformations for all groups for a total of $6^B$ combinations, where $B$ is the number of groups. The node-repulsion Linlog function [130] is used as the readability metric to find the optimum. Note that only node pairs between boxes are considered in the measurement, and we sample them randomly to compute an approximation of the function for performance speedup.

**Bundle edges.** Finally, to further reveal the pattern of connections, we apply edge bundling to the graph layout to reduce visual clutter. Out of several edge bundling algorithms [193], we choose the multilevel agglomerative one [54] for its speed and competitive visual performance. We note that it can introduce distorted edges, which are harder to trace than to unchanged ones, so we allow the user to toggle between straight and bundled edges.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>N/E</th>
<th>N total</th>
<th>E k-hop</th>
<th>k-hop</th>
<th>N box</th>
<th>B</th>
<th>UMAP (step 3)</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movie 1K/5K</td>
<td>1K</td>
<td>1088</td>
<td>5000</td>
<td>633</td>
<td>4</td>
<td></td>
<td>4.1</td>
<td>5.5</td>
</tr>
<tr>
<td>Movie 1K/5K</td>
<td>1K</td>
<td>1088</td>
<td>5000</td>
<td>370</td>
<td>5</td>
<td></td>
<td>2.5</td>
<td>4.7</td>
</tr>
<tr>
<td>Cora 3K/5K</td>
<td>3K</td>
<td>989</td>
<td>2010</td>
<td>576</td>
<td>3</td>
<td></td>
<td>3.3</td>
<td>3.9</td>
</tr>
<tr>
<td>Cora 3K/5K</td>
<td>3K</td>
<td>2116</td>
<td>4222</td>
<td>1028</td>
<td>4</td>
<td></td>
<td>8.5</td>
<td>9.6</td>
</tr>
<tr>
<td>Coauthor 10K/54K</td>
<td>10K</td>
<td>9528</td>
<td>43317</td>
<td>4254</td>
<td>3</td>
<td></td>
<td>47.1</td>
<td>57.6</td>
</tr>
<tr>
<td>Coauthor 10K/54K</td>
<td>10K</td>
<td>9951</td>
<td>44261</td>
<td>6544</td>
<td>4</td>
<td></td>
<td>110.1</td>
<td>123.9</td>
</tr>
</tbody>
</table>

Table 5.1: Computational benchmarks for the K-hop layout. N is #nodes, E is #edges: approximate for full graph and exact for used within k-hop layout. N box is maximum #nodes in any box (key dependency), B is #boxes; times in seconds.

### 5.5.2 Scalability

To aid the perceptual scalability of our approach, our emphasis is on reducing edge-edge crossings, which would hinder users’ judgement about topology and connectivity. Our choice to reveal clustering structure through the K-hop layout constitutes a trade-off, where there could be substantial node-node overlap in the K-hop layout.

For computational scalability, we instrument the code to obtain elapsed (wall-clock) timings, and measure the running times with different interactively chosen focal groups for three datasets: Movie (N=1K, E=3K) and Cora (N=3K, E=5K) as introduced in Section 5.2.2, and Coauthor (N=10K, E=54K) extracted from a larger graph of 69K nodes [166]. The experiment is conducted in Chrome on a 2020 MacBook Pro with 2.3GHz 8-core i9 CPU. We present the results in Table 5.1. In addition to the total graph size, we provide the number of nodes and edges used within for the k-hop graph layout (the combination of the focal nodes and their neighbors within k hops). We see that sometimes the k-hop focal layout incorporates all nodes and edges but sometimes shows only a subset of them, depending on the choice of focal groups.
Design alternatives to K-hop layout: (a) D3 force-directed layout with strict bounding box; (b) unbounded D3 force-directed layout with repulsion force between groups; (c) WebCola force-directed layout with constraints; (d) space-filling curve layout using spiral curve.

Graphs with 1K nodes like those in the Movie dataset take a few seconds to compute, whereas those with 10K nodes take significantly longer. The UMAP computation (step 3) is the computational bottleneck in our algorithm, as shown the timing numbers on the two rightmost columns. We note that the times do not depend on the total number of nodes, but instead, the largest number within a single bounding box, because we run UMAP processes in parallel across multiple worker threads for each box. We thus include the node count within the maximum box in the table. For example, the time for the last run (with 6544 nodes in the most numerous box) of 110 seconds doubles that of the previous run (4254 nodes maximum) of 50 seconds on the same Coauthor dataset. Compared to the UMAP computation, the time required for step 1 (divide neighbors), 2 (place boxes), and 4 (adjust group) is negligible; the optional 5th step (edge bundling) takes about 10 – 20\% of the total time.

5.5.3 Design alternatives

We tried three design alternatives to the K-HOP TOPOLOGY, which were less effective at showing neighbor hops and clustering structure than our final choice: D3 force-directed, WebCola, and space-filling spirals.

The D3 force-directed layout is versatile as we can configure the type of strength of forces. Similarly to the K-hop layout, we use topological distance as the pulling
force between nodes, instead of the usual approach of using the edges to control the forces. Figure 5.5 (a) and (b) shows two versions of this layout. The first take is bounded by boxes like the K-hop layout, but the nodes are pulled towards other groups and positioned alongside the group boundary as an undesired artifact. It does not show clustering structure within a group. The second take is not bounded strictly but has a strong repulsion force to separate nodes of different neighbor groups. The separation of hops is not obvious enough, especially for homogeneous graphs where the nodes are not shape coded.

The WebCola layout [43] (Figure 5.5c) allows us to specify positional constraints between nodes, but due to its force-directed nature, it suffers from a similar problem as the D3 version in Figure 5.5a.

The space-filling curve layout [123] is one of the fastest layout algorithms. We choose a spiral curve as it can separate nodes of different groups from center to peripheral. On a polar coordinate system, we use the topological distance as the distance on the curve between two consecutive nodes. From Figure 5.5d, we can see that it has some spatial division between different groups but is even worse than the D3 version. It also comes with an artifact that sometimes proximity in the polar coordinate system does not match the proximity in topology: nodes are placed with different radii but similar angles. Moreover, such a spiral space-filling design does not use space effectively, with a lot of unused space in the peripheral area where the less important neighbors reside.

5.6 Results

We evaluate CorGIE in two ways: we use it to address the questions in the two motivating scenarios, and we recruited two GNN experts to participate in a user study to evaluate its utility and usability.

We first present our two usage scenarios. We then describe our study design, and discuss highlights of two expert user sessions. We then summarize the study participants’ feedback.
Below, we systematically describe four aspects of CorGIE usage: the visualization task being conducted, the visual operation used within the CorGIE interface, the direct observation that is possible from the CorGIE display, and any resulting inference of authors or participants. Here we tag only the visualization task with color; in Supplemental Section D.4, we provide a version of the text with all four aspects color-coded.

5.6.1 Usage scenario 1: Movie Recommendation

To obtain an overview of the dataset, we first color the nodes by their node type (user and movie). Figure 5.6a shows that we can see two clusters of movies in blue on the right-hand side and many clusters of users in orange in the rest of the latent space. After coloring the nodes by the underlying positional colormap in the latent space, we observe in the GLOBAL TOPOLOGY view (Figure 5.3g) that most user nodes (triangles) are laid out around movie nodes (circles). Many nodes that are connected by only one edge are placed around the periphery of the layout, indicating that there are many users who only watched one movie. In Figure 5.6b, we notice that the peripheral users that are connected to the same movies have similar colors: for example, the triangles on the bottom right are all green. We can thus infer that the GNN does a good job in grouping these one-time users reviewing the same movie.

To compare user clusters, we select and focus on two clusters from the LATENT SPACE, one in the green zone (foc-0) on the left, the other in the purple zone (foc-1) on the upper right, as shown in Figure 5.3. In the K-HOP TOPOLOGY (Figure 5.3b), there appears to be three green sub-clusters in foc-0, and the green nodes connect to many nodes in the hop-1 box, while foc-1 does not show any salient internal structure. To understand the topological difference between the two focal groups, we explore their connections in the K-HOP TOPOLOGY with hover and select actions. Figure 5.6c and 5.6d show the visual highlights in the K-HOP TOPOLOGY when selecting the green foc-0 nodes and the purple foc-1 nodes respectively. We can see that the green nodes in the top foc-0 box connect to all hop-1 neighbors, but the
Figure 5.6: Check overview of the *Movie* scenario: (a) LATENT SPACE, movie nodes in blue and user nodes in orange; (b) inset of GLOBAL TOPOLOGY showing similar peripheral nodes in similar green color (closeup of lower right for screenshot in Figure 5.3). When comparing two clusters of user nodes, (c) we select nodes in foc-1 of the K-HOP TOPOLOGY, highlighting themselves and four neighbors in the hop-1 box, and also in the (d) GLOBAL TOPOLOGY. (e) We compare distances between topology and latent space, and (f) between feature and latent space.

purple nodes in the lower foc-1 box only connect to four of those neighbors. We infer that the GNN has learned the topological difference and thus separates them in the latent space.
Figure 5.7: Check movie recommendations: (a) top 5 recommended movies for user 5355; (b) K-HOP TOPOLOGY focusing user 5355 and movie LotR; (c) hover on the movie Suicide Squad that the user has watched before; (d) topological distance between 5355 and LotR is large (0.97); (e) K-HOP TOPOLOGY for user 587 and movie LotR.

Besides graph topologies, we also want to check whether distances match between spaces. In the DISTANCE COMPARISON VIEW (Figure 5.6e), we can see that the within group distances in latent space are small, while the between group distances are large, which confirms that we have picked two distant clusters from the LATENT SPACE. We notice that the topological distances within the green cluster (foc-0) are relatively large considering they belong to the same cluster (leftmost vertical histogram in Figure 5.6c), which actually matches the existence of three sub-clusters in the K-HOP TOPOLOGY. The feature distances in all three charts of Figure 5.6f are small and similar, indicating that the node features (#votes and average vote of a user) cannot distinguish foc-0 and foc-1. We further confirm the ineffectiveness of the features by reading the feature distances by picking some other node groups (not shown in figure), which indicates that they are not useful features, and could perhaps be removed from the dataset.

After exploring the clustering structure, we inspect instances of recommendation. In Figure 5.7a we select the user node 5355, and list its top 5 recommended movies. The first one is The Lord of the Rings: the Return of King (LotR). We would like to understand why GNN decides to recommend LotR, so we focus this recommendation by clicking on it. CorGIE automatically creates two focal groups with the user 5355 in foc-0 and movie LotR in foc-1 (Figure 5.7b). The K-HOP TOPOLOGY shows that user 5355 watched two movies (in pink) before. When we hover on the two pink nodes in the hop-1 box (Figure 5.7c), we find out that each only shares a few users with the recommended movie LotR; for example, Suicide Squad has only
5 hop-1 shared neighbors. We believe that this recommendation is poor, potentially indicating that we are not there yet with GNN training. We further confirmed this problem by reading the topological 0.05 and latent 0.97 distances between user 5355 and movie LotR (Figure 5.7e), which seem to be correlated negatively.

We later find a recommendation that makes sense: user 587 and movie LotR. As shown in Figure 5.7e, the two movies that 587 watched, Inception and The Dark Night, share many users with the recommended movie LotR (many hop-1 neighbors are highlighted when hovering on Inception). Moreover, the topological distance is 0.72 in this case, which is relatively small compared to the overall distance distribution (not shown in figure). CorGIE thus shows evidence that this recommendation is well supported.

We repeat this process to check other recommendations, and we find many that do not make sense. We conclude that this training result is not satisfactory. It could be due to the “cold start” problem in such a small dataset, where most users only watch 1 or 2 movies. Also, the node features were not very useful: e.g., we know that #votes cannot distinguish users effectively. To improve the recommendation, we might try a different model or fix the dataset problems.

### 5.6.2 Usage scenario 2: Cora

As with the Movie scenario, we first check the clustering structure of paper nodes. We color the nodes by the predicted labels to see if the label distribution makes sense. In LATENT SPACE (Figure 5.8a), we can see that the different classes of papers are roughly separated to different areas. In GLOBAL TOPOLOGY (Figure 5.8b), although the force-directed layout lacks much structure, we can still see that nearby nodes are in similar colors. In LATENT NEIGHBOR BLOCKS (Figure 5.8c), we can see that the red-outlined origin and its surrounding cells are darker for all the blocks. All three observations indicate that the GNN has done a good job in classifying papers using the graph topology.

Next, we inspect and compare a few clusters. For example, when we compare
(select and focus) the entire red cluster and a left part of the blue one (Figure 5.8d), the K-HOP TOPOLOGY (Figure 5.8e) shows that most nodes of the same color connect to each other, and the diff chart in the NODE FEATURES VIEW (Figure 5.8f) signals a considerable amount of different words between the two paper clusters (many visible dark strips on the diff row). These observations reinforce our good impression on the training result.

As the GNN seems to classify most nodes successfully, we look for problematic nodes and edges. We brush the distance scatterplot to highlight the node pairs with large latent distances but small topological distances (i.e. the bottom right area), as shown in Figure 5.9a, and we focus one of these problematic node pairs. In the K-HOP TOPOLOGY (Figure 5.9b), we find through a few rounds of interactive hover that the two focal nodes only share one 1st-hop neighbor (circled in red), which has many nodes connected in the second hop. As the Jaccard distance accounts for multiple hops of neighbors, the large number of shared hop-2 neighbors can explain the low topological distance (0.23). We conjecture that the GNN decides to locate them far from each other due to the large difference in the first hop. Further investigation showed that the node in foc-0 is mis-classified, which hints at the limitations of this GNN model to deal with such situations.
Figure 5.8: Explore overview of Cora dataset, with nodes colored by predicted label: (a) LATENT SPACE; (b) GLOBAL TOPOLOGY; (c) LATENT NEIGHBOR BLOCKS with zoomed-in inset showing that the red-outlined cells are darker than others. Compare two paper clusters: (d) focus on the red and blue clusters in the LATENT SPACE; (e) K-HOP TOPOLOGY; (f) NODE FEATURES VIEW with inset showing the word count differences between foc-0 and foc-1.
Figure 5.9: Find problematic node pairs in the Cora dataset: (a) select node pairs with high latent distance but low topological distance; (b) K-HOP TOPOLOGY focusing on a buggy node pair, with only one shared hop-1 neighbor (circled) and many shared hop-2 neighbors.
5.6.3 Expert study overview

One of the two GNN experts is a PhD student (P1), and the other is an industrial lab researcher (P2). Both have published multiple GNN papers in top-tier venues in recent years. We had a meeting with each participant a few months before the study when we were iterating on the tasks and design of CorGIE. In this meeting, we discussed the pain points in their workflow of model training and confirmed that our correspondence approach matches their mental model, but did not show any version of the CorGIE interface to the participants at that time. In the remote study session, I introduced and demonstrated a near-final version CorGIE for about 40 minutes. Then participants used that version of CorGIE on their own datasets for about 30 minutes, followed by a semi-structured interview to gather their feedback.

The results provide preliminary evidence of the utility of CorGIE in fulfilling the tasks we proposed in Section 5.3.3, as the experts were highly positive about the effectiveness of CorGIE.

5.6.4 Expert session 1: Cora decision boundary

In this case study, expert P1 first chose to perform quite similar tasks as in our usage scenario. Then he wanted to check the overview to study misclassification. He colored the nodes by label correctness (Figure 5.10a), where he saw that the red (misclassified) nodes are distributed across different clusters. He used a filtered brush selection to highlight and focus the wrongly predicted nodes within the middle cluster in the LATENT SPACE. He colored the nodes by the predicted labels (Figure 5.10d) and true labels (Figure 5.10e) respectively, so he could understand which classes are wrong. It appeared that the GNN predicted these nodes as the orange class similar to their hop-1 neighbors, but the ground truth labels state that they belong to multiple different classes (blue, green, red, etc) although the hop-1 neighbors are still orange. Further inspection with hover and brush selection shows that the focal nodes are loosely connected to their hop-1 neighbors, but the hop-1 neighbors are tightly connected to the hop-2 ones. Based on the variance in true
Figure 5.10: P1 explored the decision boundary in the Cora dataset: (a) he focused on the wrongly predicted nodes within a cluster (in the bounding box) in the LATENT SPACE; he compared (b) feature distances of the entire graph to (c) those within the focused nodes; he inspected neighbors in the K-HOP TOPOLOGY that are colored by (d) predicted label and (e) ground-truth label.

labels and the sparsity in topology, P1 inferred that these nodes were sitting at the decision boundary between two or multiple classes in the graph topology. It was his first time to visually inspect the boundary nodes even though he had used this dataset for years. He emphasized the importance of understanding the decision boundary if he needed to improve the GNN model.

After exploring the topology, P1 turned to DISTANCE COMPARISON for more information on the node features, where he found the feature distances within the focal nodes are relatively large compared to the overall distribution (Figure 5.10c&b). He conjectured that the current model performed as he expected, and that it would be unlikely to classify the boundary nodes correctly even if he kept training more epochs. He also conjectured that the GNN should use the node features in a better
way to correct these nodes. As he confirmed that the GNN has done a good job for most nodes except some boundary ones, he concluded that there is no need to keep training more epochs, and training should be stopped to avoid over-fitting.

5.6.5 Expert session P2: USAir

We present how P2 used CorGIE on the dataset USAir from his previous GNN paper. It is a homogeneous graph showing the air routes (2126 edges) between airports in the US (332 nodes). There are no node features in this graph. To obtain a node embedding, we used a GCN [93] to predict links between the airports in an unsupervised setting.

After CorGIE loads (Figure 5.11), P2 first checks the overview of the graph. The nodes are colored by their 2D latent space position. The LATENT SPACE VIEW shows that there are many clusters, and the one in the lower left (green) corner appears to be very different from the others. The initial global layout shows that most of the nodes connect to neighbors with the same colors, and the LATENT NEIGHBOR BLOCKS VIEW shows that the topological neighbors are placed nearby.
Figure 5.12: Find buggy node pairs. (left) Popup customization dialog to pick the node pairs that are connected in the input graph but predicted as false by the GNN; (middle) the resulting 28 erroneous node pairs shown in the scatterplot of **DISTANCE COMPARISON VIEW**; (right) the 28 error node pairs in the list on the right side of that view.

Figure 5.13: P2 focuses on the first buggy node pair (124-283), and hovers on one of the focal nodes to see its neighbors.

(the dark blocks are close to the red outlined block). We can infer that the GNN node embedding makes sense at a high level.
P2 proceeds to find buggy link predictions by the GNN. With the help of the author, he creates a filter to single out the node pairs that are connected in the input graph but are predicted false by the GNN, as shown in Figure 5.12, with the popup window that appears after clicking on the wrench icon in the upper right corner of the DISTANCE COMPARISON VIEW. He then clicks the first buggy node pair 124-283 in the list on the right side of that view, which automatically creates two new focus groups of a single node each and a new k-hop focal layout with those groups. He hovers on the focal nodes in the focal layout to check their neighbors (Figure 5.13), and he finds out that they actually share many neighbors in both the first hop and the second hop. The topological distance, as shown in the distance comparison, is 0.19, which is a very low value, which further confirms what he finds in the focal layout. However, the latent distance is 0.50, which equals the threshold in GNN that separates true and false predictions. He infers that this erroneous predicted pair is located in the decision boundary area, making it difficult to predict.

### 5.6.6 Expert feedback summary

We summarize the participants’ feedback throughout the sessions, including the questions and discussions while using CorGIE the semi-structured interview at the end. Both participants were impressed after they watched our demonstration and used the tool themselves for the first time.

**Task abstraction.** They confirmed that the two-step task framework (specify and correspond) is powerful for evaluating GNNs, especially for the iterative refinement of focal nodes, and both thought CorGIE is useful for debugging node embeddings (P1: “finding decision boundary nodes”, P2: “debug [models] quickly in an intense leaderboard competition in ML research”).

**CorGIE interface.** They stated that most views are straightforward to understand after the introductory explanation, but the heatmap and heat-strips in the NODE FEATURES VIEW are less intuitive. The three-level interaction is easy to use and suitable for the tasks. As for the usability and learnability, we observed that both
participants picked up the main functions very quickly, which the participants also verbally confirmed in the interview.

However, they criticized the interface as being visually busy/complex, making it difficult to locate a specific object at first glance. We thus improved the interface layouts according to their advice, e.g., we made the focus action menu much more salient, changing it to the eye-catching final design of a corgi paw dropping down from the top. Both expressed the excitement to incorporate CorGIE into their research workflow once it is available for deployment.

**K-hop layout.** Both participants consider the K-HOP TOPOLOGY the most useful feature in CorGIE, which validates the success of our novel K-hop layout algorithm. The representation aligns with how they would think about GNNs.

## 5.7 Related work

CorGIE relates to previous research on explainable AI, which uses visualization to explain machine learning models. In this section we relate and compare CorGIE to explainable visualizations for non-graph models, latent space, and graph neural networks. We also discuss the related work for our new K-hop layout.

### 5.7.1 Visualizations for machine learning models

Many visualization tools have been developed to explain and evaluate machine learning models, especially in the recent five years. Readers can refer to many survey papers to have a comprehensive understanding on this topic [24, 78]. Most such tools target models for images/videos and text sequences, most of which are not modeled as graphs. Although the seminal GNN work, GCN [93] stems from convolutional neural networks (CNNs) for images (as they both use convolution operation), it is not practical to simply retarget tools for CNNs to GNNs. The key difference is that pixel neighbors in an image are intrinsically different from topological neighbors in a graph. Sequence data like text in natural languages is
even more different. Unlike many visualizations that try to open the “black box” of a neural network and those that keep the “black box” completely opaque, we propose a “grey box” approach that only leverages a key concept in GNN - neighborhood aggregation (Section 5.2.1) - to balance the generalizability and specificity for GNN models.

5.7.2 Latent space interpreter

The study of the latent space, also known as the embedding space, has attracted substantial attention, especially in text-based ML practices. Embeddings eliminate the error-prone process of feature selection and they can be pre-trained for many different downstream applications [189]. As CorGIE supports exploration of the correspondences between a latent space and a graph, it directly relates to much previous work on latent space visualization.

One theme is to reveal local and global structures of one or multiple latent spaces. Ghosh et al. [55] developed VisExPres, an interactive toolkit for user-driven evaluation of embeddings. Heimerl et al. [73] compare embeddings based on different quantitative metrics, while Cutura et al. does so with dimensional reduction techniques and matrix visualizations [30]. Liu et al. [108] analogize the process of mapping and comparing semantic dimensions within latent spaces to latent space cartography. This theme mainly focuses on digging deep inside latent spaces, which is different from our approach that tries to connect a latent space to the input.

Another theme, dimension reduction (DR), is also loosely related as many non-linear DR techniques produce latent spaces [131]. There are many visual tools for DR results: some for presenting the data points [158], and some connect the DR results back to their semantically meaningful high-dimensional spaces [50, 161]. Their original input is usually tabular data, where a data item has multiple features (also known as attributes), but none of them consider graph topology like CorGIE.
5.7.3 GNN interpreter

There has been some effort in the GNN community on GNN interpretation. Most of the previous work focuses on generating algorithms or models to conduct feature and neighbor analysis \[190\]. Huang et al. \[81\] propose GraphLIME to produce a few most representative features in the neighborhood of a node. One of the most well-known work is the GNNExplainer \[188\] by Ying et al., who propose a method to compute the most important nodes of one or a group of user-specified nodes based on information theory. Rao et al. \[147\] follow up on GNNExplainer and propose xFraud to target at fraud detection specifically. Pope et al. \[137\] extend explainability algorithms for CNN like saliency map, class activation mapping, and back-propagation to Graph-CNN. This thread of work does not sufficiently support human-in-the-loop visual exploration, and so falls short in providing an overview of how well a GNN learns from the input graph and in connecting the exemplar inspections iteratively.

Two recent papers visualize graph models. Li et al. \[106\] propose EmbeddingVis to compare multiple graph embeddings generated from different models. It focuses on how the node metrics (e.g. degree, centrality) are preserved in the embeddings, but does not directly support topological neighborhood exploration nor node features. As this research project was conducted before the recent trends of graph neural networks that use neighborhood aggregation, their embeddings are not generated by GNNs but other graph models. A paper by Jin et al. \[89\], who developed GNNVis to diagnose errors in GNN, is the most similar related work to CorGIE. It only targets one downstream ML application, node classification, which limits its scope of usage. Unlike CorGIE, GNNVis ignores the latent space and takes another route: it supports finding errors in the classification predictions, and comparing against two surrogate models to approximate the erroneous component in GNN.
5.7.4 Graph layout

We discuss the related work for our technical contribution, the K-hop layout. A graph layout survey by Gibs et al. [56] categorizes graph layouts into three approaches: force-directed layouts, dimensional reduction layouts (e.g., MDS, t-SNE, UMAP), and multi-level graph layouts (see survey by McGee et al. [117]). Our K-hop layout combines the DR and multi-level approaches. Out of dozens of layout algorithms, ours is most related to those focusing on clusters. IPSep-CoLa [43] by Dwyer et al. is a force-directed layout specializes at separation constraints. Our investigation of its utility for this setting shows that it generates undesired artifacts (Figure 5.5c). Our K-hop layout is similar to the Group-In-a-Box layout proposed by Rodrigues et al. [152] for category-based partitions of social networks. It uses the space-filling treemap techniques to separate the clusters, whereas our division into groups is dynamic based on user selections and combines DR techniques. The LinLog layout [130] proposed by Noack is an energy-based model for cluster separation, and it inspires our readability metric in the adjustment step.

5.8 Discussion and future work

On reflection, after the whole process of characterizing the problem, designing the interface, and evaluating it, we believe that the high-level idea of exploring correspondences among input, output, and internal data structures is useful for GNN interpretation and may be generalizable to other deep neural networks. As a neural network itself is usually not interpretable for many reasons (e.g. non-linearity, layering), the approach to “open the black box” completely demands substantial complexity of interpretation, and demands high expertise from the users. We prefer a “grey-box” approach balancing the exposure of internal structure to a nonzero but minimum level. Though often not exposed to the end users for interpretation, the node embedding is universal to all GNNs. By finding correspondences between it and the input data (graph), we can infer if the GNN has achieved satisfactory results without fiddling around the internal structure of a neural network. We imagine this approach could be generalized for other types of neural networks.
We would also like to promote our notion of data spaces. Our data abstraction consists of the three data spaces – latent, topology, and feature – alongside a task abstraction that connects them. This mental model of connecting data spaces helps us design the views: some are dedicated for a single space, while some connect multiple spaces. It also contributes to the usability and learnability of CorGIE, for which we have preliminary evidence through the feedback from two GNN experts. It would be interesting future work to introduce more data spaces, such as a geospatial space to deal with GNNs that specialize in geospatial data [66].

It would be useful future work to handle larger graphs. Due to the K-hop layout computation, the current version of CorGIE cannot guarantee a smooth interactive user experience for graphs with more than 20K nodes. Although many popular benchmark datasets are within this scale, the ML community is moving forward with larger datasets, such as those on the OGB platform [80].

To further extend CorGIE, we could incorporate the algorithms like GNNExplainer [188] either in the K-HOP TOPOLOGY VIEW implicitly or in a dedicated view explicitly. This idea was also independently brought up by one of the GNN experts. We could also support more ways to specify nodes, such as topological statistics. Finally, we could enable the comparison of multiple node embeddings of the same input graph, to help evaluate model re-architecting and hyper-parameter tuning. A future paper could study how to enable GNN developers to explore the correspondences between multiple input graphs and their node/graph embeddings.

### 5.9 Summary

In this work, we present a task abstraction for exploring the correspondences between an input graph and the latent space created by a GNN, to understand if GNN has learned important characteristics from the graph and to find bugs in the latent space. Based on this abstraction, we develop an interactive multi-view tool, CorGIE, which is validated through usage scenarios and case studies with GNN ex-
perts. As the most important component in CorGIE, we propose the K-hop graph layout to reveal how GNNs aggregate information for nodes of interest. Both case studies and expert studies validate the effectiveness of bringing CorGIE into a GNN model development life-cycle. We envision that our novel data and task abstraction, in conjunction with our design rationales and implementation considerations, could serve as a stepping stone for future researches.
Chapter 6

Reflection and Conclusions

In the previous chapters, I introduce four projects with a common theme of visualizing multi-level structures. In this final chapter, I will first discuss the limitations of each project, then present my reflections, before concluding my PhD work.

6.1 Limitations

In ADVIEW, our design can handle datasets of hundreds of trees with about 100 leaf nodes per tree, which is limited by the task of showing the full detail of a reference tree dendrogram on a single screen without scrolling. We do not need to worry about the breadth and depth of a tree, as they are bounded by the number of nodes: an extreme case could be a ladder-like tree, where there are only left or right descendants for each internal node. For the number of trees, there is no hard limit, but our current implementation on the browser can only deal with roughly 500 trees at interactive frame rates. It is possible to exceed these limits by closing the Individual AD view as the number of SVG elements is reduced drastically.

In SPRAWLTER, we only consider readability metrics. It would be useful to investigate how our metrics can combine with others to make better graph layouts. In
terms of computational complexity, our current implementation only adds a con-
stant multiplicative factor to the count-based approach, which makes it almost as
applicable as the baseline.

In LOGSEG, the simple SVM model is fast to train and use. It has a small number
of parameters compared to other machine learning models like neural networks.
The primary concern for such a model is underfitting. User behavior in software
could vary across different people or even different sessions of the same person. A
human will make mistakes, which could mislead the model. Users interleave dif-
ferent subtasks together, rendering the assumption of modeling user behavior with
event sequences problematic. There could be many unpredictable factors when
using software, such as interrupting the workflow by a colleague. Our segmenta-
tion model might be unlikely to perform correctly in these complicated cases. For
scalability, there are no limits on the number of events in a sequence as this online
algorithm can work as the user uses Photoshop. The depth of the segmentation
hierarchy depends on the nature of the user’s workflow. Usually it will be a small
number as the best hierarchical segmentation reflects the user’s mental model about
his workflow.

In CORGIE, we limit the scope to one input graph and one node embedding only.
There could be many future directions: GNNs that take many graphs as input and
generate graph embeddings, GNNs that output edge embeddings, and comparison
of multiple embeddings. The current implementation of K-hop layout also has
limits on the number of nodes, that is, less than 20K nodes, to maintain a smooth
interactive speed on a browser.

6.2 Reflection on multi-level structures

Multi-level structures are the common components in the four research projects.
Multi-level structure is a broad and general concept, not a specific technique. The
structures exist in almost all data types in information visualization: they could
be either hierarchical or non-hierarchical; they could be tables, trees, graphs, time
series, text, etc. This concept is applicable to many problem domains. My own
examples include phylogenetics in biology, user engineering in image editing software, graph neural networks in machine learning. I have also seen many examples in the literature, such as turbulence in scientific visualization [129] and human resource management [148]. There are various techniques to extract multi-level structures depending on the domain, data and task. For example, one can use a general hierarchical clustering algorithm with a distance metric to measure differences between data items, or, simply categorize the items according to a categorical data attribute. Sometimes these structures are not even computed with algorithms, but manually specified by users in cases like CORGIE. It is hard for me to imagine an overarching framework for computing multi-level structures in data.

The visualization of multi-level structures is not a trivial problem in the design layer of the nested model. The hierarchical nature dictates the need of different views for different levels in order to solve a domain problem that contains multi-level tasks. In ADVIEW, the interface contains views for all levels in a tree collection dataset, and some of the levels (e.g., subsets of trees) are shown in multiple views (e.g., the tree distribution view and cluster aggregated dendrogram view), which is detailed in Supplemental Section A.1. It is not enough to have multiple static views. Following the guidelines for view coordination in the visualization literature, such as Qu et al. [145], we link the views with consistent visual encoding and user interactions. In ADVIEW, the color palette is carefully designed to match the semantics, where the five medium-saturation colors (Figure 2.10) for the subtree highlight blocks have roughly equal luminance so that no focal subtree seems more important than the others. Also, the color palette is consistent between the reference tree and all the aggregated dendrograms so that users can quickly catch their connections. In CORGIE, the shape, color, and size of nodes are consistent across different views.

User interaction is also crucial for linking views. It is important to design how the interface reacts to user actions such as clicking and hovering, usually with changes in the visual encoding. Meanwhile, it is also important and challenging to maintain the consistency of interface reaction to the same user action on different views. In CORGIE, we design three kinds of user interactions from light-weight to heavy-
weight: hover, select, and focus. The trigger action is consistent across views: mouse over, mouse click, and dedicated buttons. The reacting visual prompts are also consistent: strong highlights with a half-transparent mask on hovering, similar but persistent visual highlights on selection, and new graph layouts on focus.

### 6.3 Reflection on collaboration

All contributions in this dissertation are the result of collaborative work. I believe visualization for human-in-the-loop data analysis requires collaboration with stakeholders, especially domain experts (users). In *ADVIEW*, I collaborated with a group of phylogenetic biologists in UBC, who became the users of ADView. In *SPRAWLTER*, I collaborated with a visualization researcher from another university, who brought new insights to our project. In *LOGSEG* and *CORGIE*, I collaborated with industrial visualization researchers from Adobe and Uber respectively, who provided pointers to Photoshop developers or users, and machine learning developers.

Through these collaboration, I realize that there is a spectrum for the possibility of user collaboration from a tight connection with collaborators to a loose one. In *ADVIEW*, Shing Hei Zhan, a PhD student in biology, worked with us mainly through weekly meetings all along the whole research life cycle. He participated in iterations of domain characterization, data and task abstraction, design, evaluation studies, and paper writing. This tight collaboration is best for ensuring the utility of visualization tools as the collaborating domain expert understands the domain problem well. However, this could be a rare case, as it requires significant investment from the domain experts. There is also a potential danger: the scope of the project and resulting tool could overfit to one or a small group of people, leaving it less generalizable to a bigger scope and similar usage scenarios.

The *LOGSEG* and *CORGIE* project lie on the loose side of the collaboration spectrum, where none of the coauthors are domain experts. I believe this is a more usual case given the challenges to maintain a long-term tight collaboration. I started the *LOGSEG* project during my industrial internship in Adobe Research, where
Zhicheng (Leo) Liu was my mentor. He brought the resources in Adobe as we were working on Photoshop, the flagship product of the company. We talked to managers, researchers, developers, and users of Photoshop to understand the data and tasks and define the project scope. After we decided to work on segmentation of user action logs, we recruited Photoshop experts as participants to collect their usage data on Photoshop in remote study sessions. During this project, I learned to use Photoshop myself through online tutorials and have used it in my daily life since then. I, as the first author, became an amateur user of our target domain, which lowered the necessity to collaborate with Photoshop experts closely throughout the project cycle. Besides, many research artefacts were intuitive to us. For example, we could understand the process of making a poster or touch up a human face in Photoshop, even though we might not be able to do it ourselves.

In CORGIE, Yang Wang played a similar role as Leo in LOGSEG. We discussed potential research questions with his manager in Uber Research, before deciding to work on interpretation of graph models. Then we talked to graph machine learning experts in Uber to understand what graph neural networks are and their painpoints in the evaluation of GNN. We recruited two other experts for tool evaluation after we had a stable version of the tool in the late stage. In between, we reviewed many papers published in the machine learning venues and articles/blog posts by experts to ensure that what we did matched what the users want and how they think. We were able to do this in this project because we have computer science backgrounds and basic knowledge of machine learning, but this approach would not have worked in ADVIEW where a deeper knowledge of phylogenetics was required.

6.4 Reflection on research topic choices

The four projects reported in this dissertation may appear to be very different in terms of type and domain. As shown in Table 6.1, ADVIEW and CORGIE covers all layers in the nested model, while the other two do not. ADVIEW, LOGSEG, and CORGIE are problem-driven work to address concrete domain problems, while SPRAWLTER is a technique-driven work to improve a visual encoding. The four
projects span three different domains: phylogenetics in biology, user engineering in software, and graph machine learning. However, there still exists a common theme, multi-level structures, to connect them.

I chose to explore broad topics on purpose from an early stage of my PhD program, with the goal of experiencing and learning as much as possible. This choice had a negative impact on the duration of my PhD, as I had to learn about multiple new application domains over my time in the program. However, the positive impact on the breadth of my research outweighs the negative impact on duration. I strengthened my ability to learn new knowledge in unfamiliar disciplines, increased my experience of collaborating with people, and learned how to conduct different types of visualization research. The diversity in topics and types of research in this early stage of my career will greatly benefit my future career in academia.

### 6.5 Concluding remarks

I present four projects for visualizing multi-level structures in data. I summarize their coverage of the four layers of visualization concerns using the nested model in Table 6.1. To conclude, I contribute novel data and task abstraction, visual techniques, and tools to deal with multi-level structures in three domains. To solve the multi-level tasks with complex data of the target domains, the multi-level structures are either exploited directly or derived from the raw data, which are then visualized using multiple views. The abstraction and visual techniques are also benefited

<table>
<thead>
<tr>
<th>ADView</th>
<th>Sprawlter</th>
<th>LogSeg</th>
<th>CorGIE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>Phylogenetics</td>
<td>Image editing</td>
<td>Graph machine learning</td>
</tr>
<tr>
<td><strong>Data &amp; task</strong></td>
<td>Compare many trees</td>
<td>Segment logs</td>
<td>Explore correspondences</td>
</tr>
<tr>
<td><strong>Encoding design</strong></td>
<td>Aggregated Dendrogram; Multi-view tool ADView</td>
<td>Readability metrics for node-link graph; area-aware; Sprawlter</td>
<td>K-hop graph layout; multi-view tool CorGIE</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Find corresponding branches</td>
<td>Find node and edge crossings</td>
<td>Determine chunk boundaries</td>
</tr>
</tbody>
</table>

Table 6.1: A summary of contributions of my four projects covering the four layers in the nested model.
from the multi-level structures compared to previous work.

Multi-level structures can be found in many real-world problems, and they are becoming more common and important due to the growing complexity of data and tasks. There remain many research opportunities on the usage of multi-level structures to solve concrete domain problems.
Bibliography

[1] Newick tree format. 
http://evolution.genetics.washington.edu/phylip/newicktree.html \(\rightarrow\) pages \[36, 41\]


174


177


178


186


Appendix A

Supplemental Material for ADView

A.1 View coordination details

Table A.1 documents the view coordination discussed in Section 2.6.6, showing which aspect of the data is visually encoded across all five levels of detail for each of the eight views. The table has six columns since we break out branches from leaves for clarity; both of these are the lowest level of detail. We duplicate Figure 2.7 and put it here for a quick reference showing all of these views, as Figure A.1.
<table>
<thead>
<tr>
<th>View</th>
<th>Level of detail (LoD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree collection</td>
<td>Subset of trees</td>
</tr>
<tr>
<td>Individual tree</td>
<td>one AD block</td>
</tr>
<tr>
<td>Subtree</td>
<td>line or collapsed</td>
</tr>
<tr>
<td>Branch &amp; its attributes</td>
<td>text label &amp; black dot</td>
</tr>
<tr>
<td>Leaf node</td>
<td></td>
</tr>
<tr>
<td>Reference Dendrogram</td>
<td>row segment in row</td>
</tr>
<tr>
<td>Tree Distribution</td>
<td>whole view</td>
</tr>
<tr>
<td>Cluster AD</td>
<td>one cluster</td>
</tr>
<tr>
<td>Individual AD</td>
<td>one AD block</td>
</tr>
<tr>
<td>Pairwise comparison</td>
<td>consensus tree butterfly layout</td>
</tr>
<tr>
<td>Tree Similarity</td>
<td>t-SNE scatterplot</td>
</tr>
<tr>
<td>Tree List</td>
<td>text label</td>
</tr>
</tbody>
</table>

**Table A.1:** View coordination of visual encoding across all levels of detail and all views.

**Figure A.1:** A screenshot of ADView as a reference to Table A.1
A.2 Cluster AD gradient coloring

When grouping topologically identical individual ADs into a cluster AD, we use one of the AD layouts as a proxy for the cluster AD, but use gradient color to convey the uncertainty of the proportion of taxa instead of the solid color in an individual AD. The percentage of the color fill represents the proportion of highlighted taxa in an AD block, which is fixed for individual AD but typically covers a variable range within a cluster AD. As shown in Figure A.2, block B has different proportions of orange color in the four individual ADs on the left, which can be considered as a distribution [55%, 50%, 60%, 70%]. In the cluster AD on the right, the orange in block B has a fuzzy edge, which is perceived as the variance or uncertainty of the color proportion.

We encode the complementary cumulative distribution function (c-CDF) of the percentage of color fill with color saturation to achieve the fuzzy visual effect, which was first introduced as density strip by Jackson [88]. We do not choose the probability density function (PDF), which is usually rendered as a histogram, nor the cumulative distribution function (CDF), because both of them could be misleading when mapped to color saturation in our context, as illustrated in Figure A.3.

Figure A.2: Illustration of gradient coloring in a cluster AD. The percentage represents the proportion of highlighted taxa in a block, and is encoded as a solid color in the individual AD, but gradient color in the resulting cluster AD.
Figure A.3: Illustration of different encoding representation for the same normal distribution (mean=50%, standard deviation is small). Color saturation is mapped to the value of the probability function shown inside the block. Probability functions from left to right are the density function, the cumulative distribution function, and the complementary cumulative distribution function.
A.3 Algorithm details

We supplement more detail about the aggregated dendrograms generation and rendering (Section 2.7) for the purpose of replication.

A.3.1 AD layout function parameters

Figure A.4 illustrates the parameters used in generating AD layouts. We use a best-effort mechanism to adapt the AD layout to user-specified resolution, described in Section 2.7.2. If an AD layout does not pass our legibility test, we will shrink some of the flexible parameters and re-generate a layout.

There are two cascading sets of flexible parameters that can be changed: 1) the number of context levels, a metric to control how many context blocks to show; 2) inter-block gaps, branch lengths, block sizes.

A.3.2 Front-end caching for rendering ADs

The rendering of aggregated dendrograms is handled by the front end; that is, the browser. It is not a trivial process, as shown in Figure A.5 and there might be hundreds of ADs to compute. Therefore, to achieve reasonable response time to user interactions, especially for frequent ones, such as hovering over an AD, we cannot afford to run the whole pipeline from start to end.

Notice that many frequent interactions do not affect the layout of ADs: for example, hovering only draws a black border around the AD, and resorting only affects the order of ADs. We can cache the intermediate results, namely, the ovals in Figure A.5, so that a user interaction only triggers re-computation of the necessary steps reusing some of the cached results.

We use ReactJS\(^1\) and Redux\(^2\) as our rendering and interaction pipeline, and the

\(^1\)https://reactjs.org/
\(^2\)https://redux.js.org/docs/introduction/
Figure A.4: Illustration of parameters in an aggregated dendrogram layout.

The memoization functionality provided by the Reselect library[^3]. The memoization keeps an internal mapping between the state of data elements specified by the developer and the computation results such as AD layouts. It detects if the state changes, that is, if the state object is different from the previous one, and determines to whether reuse stored results or trigger a re-computation. In Figure A.5, selecting a subtree in the reference tree leads to a re-computation of the whole pipeline because it changes the corresponding branches; changing the sorting order of AD will reuse the results of filtered AD layouts.

[^3]: https://github.com/reactjs/reselect
Figure A.5: Pipeline on the frontend to render the aggregated dendrograms. The results in ovals are cached so that certain kinds of frequent user interaction can take place without triggering re-computation of intermediate results that remain useable.
Table A.2: Relevant information of participants and datasets used in the study.

### A.4 Expert Interview Study

#### A.4.1 Participants

Here, we summarize the relevant information of all participants and their datasets we recruited in Table A.2.

#### A.4.2 Interview Questions

1. Before today, how long have you been analyzing this dataset and with what tools? Did you generate it yourself or get it from someone else?

2. Did you find any interesting biological insights in this dataset?
   
   (a) Could you confirm things that you already knew? And how long did it take to see this compared to other tools?
   
   (b) Did you notice anything new in this dataset?

3. What capabilities of the tool are useful for your research?

4. Is there any functionality that is missing from the tool that would be useful?

5. Are there aspects of the tool that are confusing, misleading or awkward?
6. In our corresponding branch matching algorithm, we currently compute the similarity metrics between every pair of branches and find the most similar branch in the tree to the one in the reference tree.

(a) Does this computation make sense to you? How closely does it match your mental model of how you compare two trees? Is there some other way of thinking about support and conflicts to a specific clade in the reference tree that is an alternative to this kind of matching?

(b) Our corresponding branch matching algorithm assumes that all trees have a root. We know this assumption breaks in many cases, including when the outgroup taxa are missing or you have a non-monophyletic outgroup. Here’s an example of a tree where the root is wrong and you can see that the matching is very messy. Which clade do you think is the real match to A here? Or do you think that this question doesn’t make sense and we shouldn’t even try to find a match in this case?

(c) Do you think we should differentiate between the case where two clades are very similar (although not exactly the same) from the case where two clades are very different? At some point is it no longer useful to show a ”best match” past some cutoff value of “too different”?

7. Does the visual design of AD make sense to you and match your mental model of how you think of a phylogenetic tree? The same problem with incorrect rooting affects the visual layout of the ADs. Do you think an AD with incorrect rooting could mislead you?

8. Does the tree distribution view make sense to you and match your mental model of how you think about agreement and conflict between clades respectively?
   The tree distribution view sometimes has a very long tail with many sets that contain only a single tree. Is seeing all of these tiny sets helpful in terms of conveying any interesting information to you, or would it be better to leave that out? Do you have thoughts about whether it’s conveying information about the true biology, or about systematic errors in tree reconstruction, or
whether it could just be an artifact of the matching computation in our own software?

9. Do think this tool (or an improved version of it) could be helpful for other scenarios such as horizontal gene transfer? What kind of scenarios do you think would be suitable for this tool? How often would you find a tool like this useful in the work that you do? (For example, every day, once a month, once a year . . .?) When in a project life cycle might you use something like this - at the beginning of a new project, after a specific phase of it, just before paper writing . . .?)

10. Do you have other comments about the tool?
Figure A.6: Full screenshot of ADView to explore four clades. After selecting four focal clades: A (blue): LAND PLANTS (LP), B (orange): ZYGNETOPHYCEAE (ZYGN), C (green): CHARALES (CHAR), D (red): COLEOECHAELES (COL). We compared the cluster ADs with the previous hypotheses to investigate which one (orange or green or red group) is the sister of LAND PLANTS.

A.5 Full Screenshots of Usage Scenario 1: 1KP pilot study

Figure A.6, Figure A.8, Figure A.9, and Figure A.11 are the full screenshots of the usage scenario in Section 2.8.3. We explored the two research questions in the 1KP pilot study [183] in ADView.

A.5.1 Sister group of land plants

Figure A.6, Figure A.8, Figure A.9, are full screenshots to supplement the partial screenshots in Section 2.8.3.
Figure A.7: Support for the most popular hypothesis (LP + ZYGN). We checked the distribution of support values for the most popular hypothesis (LP + ZYGN) by selecting the trees that have the LP + ZYGN clade and created a sub-collection out of them. In the Corresponding Branch view, some trees have low support values for this hypothesis, which may render some doubts on whether LP + ZYGN is truly strongly supported by this dataset.
Figure A.8: Pairwise comparison with the conflicting trees. We selected the conflicting segment in the second row (B) in tree distribution (where trees shown in brown background) and made a consensus tree, which is being compared head-to-head against the reference tree in full details. We presented the markers for the selected trees in the butterfly dendrograms, which showed an outlier species SPIROGYRA is not included in the orange (ZYGN) group. Notice that selected trees are highlighted across multiple views in the interface.
Figure A.9: Combination of user-specified clades. We combined LP and CHAR into a user-specified taxa group. This feature is used for exploring hypotheses that are not presented in the reference tree, for example that there is no single subtree that consists of LP and CHAR in this reference tree. The second cluster we selected represent the subset of trees that group LP and CHAR together (solid border indicates exact matches). With this, we checked the support values for their corresponding branches (the overarching branch above LP + CHAR) in the corresponding branch attribute view on the bottom right. We found that there are a lot of low-support corresponding branches. In other words, some trees are not certain on grouping LP and CHAR together although they appear so.
Figure A.10: Exploration of low-support branches. We noticed that some trees have low support values (below 0.5), so we selected these trees using the Corresponding Branch Attribute view, shown in the black circle at the bottom right. By displaying the tree names above the ADs, we found that most of them are generated with the “Supertree” method, which might be worth investigation later.
A.5.2 Early diversification of land plants

In Figure A.11, it is easy to find out that the monophilies of the four early lineages of land plants (Hornworts, Mosses, Liverworts, and vascular plants) are strongly supported because almost all trees agree with the reference tree in the Tree Distribution view. The widely accepted hypothesis Lv-basal in Figure A.12, that is, Liverworts is the sister-group of all other land plants, is rejected by most of the trees in this dataset. We notice that only two trees (the 4th and 6th cluster AD) support Lv-basal.

According to the 1KP paper, the widely accepted view that Liverworts, Mosses, and Hornworts are, respectively, successive sister groups to vascular plants, are not recovered in this dataset. In ADView, we can see that there are no such trees presenting this topology: (blue, (orange, (green, red))), which is a direct evidence to support the statement in the 1KP paper. The first cluster AD is exactly the Hw-basal hypothesis in Figure A.12 and the second cluster AD is compatible with the Bryo monophyletic group hypothesis.

Biologists can then connect this evidence with their domain knowledge such as what substitution models are used to generate these trees and analyze their pros and cons. ADView presents the relevant information to them, in hopes that they interpret the visualization with biological judgment.
**Figure A.11:** Screenshot of ADView exploring early lineage of LAND PLANTS. A (blue): HORNWORTS (Hw); B (orange): MOSES (Mo); C (green): LIVERWORTS (Lv); D (red): VASCULAR PLANTS (VP).

**Figure A.12:** Previous hypotheses about early diversification of LAND PLANTS. Figure excerpted from the 1KP paper [183].
A.6 Usage scenario 2: TreeFam

TreeFam [154] is a database of animal gene trees built from the genome sequences of representative animal species. The data set spans the entire evolutionary history of the phylum ANIMALIA. In this case, we utilized the TreeFam data to demonstrate how ADView may be used to discover gene trees that are concordant with well-established evolutionary relationships. More specifically, we examined and identified gene trees consistent with two widely accepted splits in animal taxonomy: PROTOSTOMIA and DEUTEROSTOMIA [168]; ECDYSOZOA and LOPHOTROCHOZOA [4]. Both of these views have received support from multi-gene phylogenetic and phylogenomic studies [75 134].

Since the initial publication in 2006, there were several updates to TreeFam. Here, we used the latest release (4.0), which includes genome sequence data from 108 animal species and one outgroup plant species (ARABIDOPSIS THALIANA). We downloaded the individual gene trees as well as a species tree that captures major evolutionary relationships in ANIMALIA from [http://treefam.genomics.org.cn/](http://treefam.genomics.org.cn/). Because ADView cannot yet handle duplicate genes, we excluded gene trees with duplicate genes. Also, we only included gene trees with at least 20% of the taxa represented. A final set of 1,317 gene trees was taken as input to ADView.

Note that we collapsed the Individual AD view when we were only looking at the Cluster AD and the Tree Distribution view, because there are more than 500 individual ADs. Using only the Cluster AD view led to much better responsiveness to user interaction because there were so many fewer visual elements to handle. Also, this choice kept the full screenshots from being extremely long.
A.6.1 **PROTOSTOMIA and DEUTEROSTOMIA**

*Figure A.13* and *Figure A.14* illustrate how the user can use ADView to visually confirm that many of the gene trees are consistent with the classical taxonomic thought that PROTOSTOMIA and DEUTEROSTOMIA are the two major monophyletic branches of BILATERIAN animals. As illustrated in *Figure A.13*, by selecting the two clades as A (PROTOSTOMIA) in blue and B (DEUTEROSTOMIA) in orange, we found subsets of gene trees entirely or partially consistent (i.e., with missing taxa) with the species tree. The evolutionary relationship between A and B is reflected in the clustered ADs shaded with brown backgrounds. We also observed several other clustered ADs in which we did not find any support for RQ3. The first clustered AD (leftmost) contains 751 gene trees having only taxa from B; similarly, the fourth clustered AD contains 77 gene trees having only taxa from A. This situation immediately reveals genes unique to either A or B, due to a biological process (clade-specific gene gain or loss) or incomplete sampling (imperfect data collection). The other cluster ADs capture seemingly discordant gene trees; however, further exploration of the gene trees under the Individual Aggregated Dendrogram view and pairwise comparison of gene trees, as shown in *Figure A.15* and *Figure A.16*, revealed that the discordant signals likely resulted from inadequate post-processing of the gene trees. The TreeFam pipeline did not remove “rogue” taxa, which are outlier taxa spuriously inserted into the wrong clades. TreeFam was assembled a decade ago, therefore the results of the pipeline do not reflect the best practices in phylogenomics today. Removal of rogue taxa (e.g., using Rogue-NaRok [2]) should eliminate most of the discordant gene trees.

Additionally, we checked whether PROTOSTOMIA and DEUTEROSTOMIA are sister clades; that is, whether A and B form a monophyletic group. By selecting C (*Figure A.14*), we found most of the gene trees (1,163) to be concordant with the monophyly of PROTOSTOMIA + DEUTEROSTOMIA. We further explored the gene trees under the Individual AD view (not shown), and found that the discordant gene trees might be caused by a variety of reasons (e.g., rogue taxa, incomplete sampling, or stochastic and systematic errors in phylogenetic methods).
Figure A.13: The major clades **PROTOSTOMIA** (A) and **DEUTEROSTOMIA** (B) are selected. This view provides overall gene tree support for the monophyly of **PROTOSTOMIA** and that of **DEUTEROSTOMIA**. The supporting clustered ADs are highlighted with brown backgrounds. The *Individual AD* view is collapsed because we were not focusing on any individual trees at this point. Collapsing the *Individual AD* view also results in faster response because there were many fewer elements to render and keep track of for the browser.
Figure A.14: Screenshot of ADView for exploring **Protostomia** and **Deuterostomia**. Both **Protostomia** and **Deuterostomia** are selected as a single group (C). This combination reveals most of the gene trees are consistent with the monophyly of **Bilateria** animals (C).
Figure A.15: We selected the discordant trees excluding trees that are missing PROTOSTOMIA or missing DEUTEROSTOMIA, and created a sub-collection, that is, a subset of the tree collection. We wanted to find out why these trees do not behave as expected.
Figure A.16: Pairwise comparison is helpful in locating the potential causes of the discordance. The orange outlier at the top shows that B has a rogue gene in the gene tree (right dendrogram) compared to the species tree (left dendrogram).
A.6.2 Ecdysozoa and Lophotrozoa

Next, we performed the same tasks, but we revisited the well-established ideas of Ecdysozoa and Lophotrozoa being the two major monophyletic branches of Protostomia. We selected A (Ecdysozoa) and B (Lophotrozoa), but found only small clustered ADs (highlighted in brown background) consistent with their monophyly, as shown in Figure A.17. We then selected only Ecdysozoa or Lophotrozoa separately. When we selected only Ecdysozoa (A), we found strong support for its monophyly, corroborated by most of the gene trees, as shown in Figure A.18. When we selected only Lophotrozoa (not shown), however, we observed no strong support in TreeFam for its monophyly, probably because there are too few representative taxa included to yield robust phylogenetic signals. Indeed, later studies involving more Lophotrozoa taxa produced strong support for its monophyly [75, 134].
Figure A.17: Screenshot of ADView for exploring ECDYSOZOA and LOPHOTROZOA. The major clades ECDYSOZOA (A) and LOPHOTROZOA (B) are selected. The clustered ADs in support of their monophylies are highlighted with brown backgrounds.
Figure A.18: Cluster ADs showing distributions of ECDYOZOA. Only ECDYOZOA (A) is selected. The clustered AD, which contains most of the gene trees, corroborating its monophyly is highlighted with a brown background.
Figure A.19: Screenshot of the tool developed by Bremm et. al. [19], on the 35-tree dataset from their paper.

A.7 Screenshots of Information Density Comparison
Figure A.20: Screenshot of ADView with the same dataset at the same screen resolution, with three subtrees selected. Information about exactly what taxa their domain experts explored is not available from the paper, so we randomly picked three subtrees.

Figure A.21: Screenshot of the tool developed by Bremm et. al [19]. Elements with similarity score below 0.5 are filtered out.
A.8 Case Studies

A.8.1 Full Screenshots of Case Study 1

Figure A.22: P1 was exploring the position of the T10 taxon with regard to a bigger monophyletic group: the direct parent of the four focal subtrees, namely T10 itself (blue) and three sibling groups (orange, green, red). The position of the blue T10 group varies; there are many cluster ADs with a relatively small number of trees in each. She also observed the same by skimming the individual ADs.

A.8.2 Full Screenshots of Case Study 2
Figure A.23: P2 identified an interesting taxon \textit{BLASTOCYSTIS HOMINIS}, shown as the orange group, that usually lives in pigs but was placed close to human related strains.
A.8.3 Case Study 3

P3 is a frequent collaborator of our domain expert co-author, and they were working together on a dataset comparing 1 species tree against 71 gene trees of 115 red algae. During the study, they were able to locate some misbehaved missing taxa in the reference tree and outliers in the gene tree, but could not continue the analysis further due to a missing engineering feature of the interface (selecting a tree by its name). After we added that feature, they used ADView independently for several weeks and made some interesting biological discoveries.

They first compared four species trees created with different methods and kinds of sequences, mainly using the *Pairwise Comparison* view, to identify the one that is most consistent with the current literature (T3), as shown in Figure A.24. Next, using that best tree among the four as a reference, they compared it against 155 gene trees built from amino acid sequences. They sought artifacts, such as unusually long branches, and anomalies, primarily “rogue taxa” where outlier leaves are spuriously inserted into the wrong subtree. They quickly scanned over the gene trees to get some insights about where discrepancies typically arise (T4 and T5). Unsurprisingly, discrepancies seem to be concentrated at deep nodes (e.g., CYANIDIALES), because in many genes the information needed to infer deep relationships has been eroded or lost. They also observed that a three-taxa subclade of POLYSIPHONIA, has undergone exceptionally high molecular evolution in a gene “rpl21”, as shown in Figure A.25. The conflict between rpl21 gene tree and the reference tree on the POLYSIPHONIA subclade is an alert to be cautious when interpreting signals from POLYSIPHONIA, well-known for uncertain phylogenetic placement. Their future work is to aggregate the results of the molecular evolution analysis across the 155 genes, and they plan to check the gene trees in ADView for details that may affect their interpretation of the results.
Figure A.24: P3 and our biologist co-author first compared four species trees inferred from different methods and different kinds of molecular sequences in order to choose the best one for later analysis.

Figure A.25: P3 and our biologist co-author used the best tree from the previous step as the reference tree, and compared it against 155 gene trees. They found the blue group is inconsistent across gene trees. An example (gene “rpl21”) is shown in the figure.
A.9 Screenshots of Dataset Upload

Users can upload their own datasets through a dedicated tab in the client browser interface. There are two steps to upload a dataset: upload the tree files, as shown in Figure A.26, and specify an outgroup if the trees are unrooted, as shown in Figure A.27. The server then performs indexing, tree distance calculation, and corresponding branch matching, as shown in Figure A.28.

Finally, the newly uploaded dataset is added to the list of all uploaded datasets, available through another tab.
Figure A.26: The first step of uploading a dataset: specify the files for a reference tree and a collection of trees.
Figure A.27: The second step of uploading a dataset: select an outgroup by either checking the names, clicking on the branches (as shown here), or paste a text file.
**Figure A.28:** Server is pre-processing data.
Appendix B

Supplemental Material for Sprawlter

B.1 Parameter analysis

We present the analysis for two parameters in the penalty mapping functions that are defined in Section 3.4.3:

\[ f_{v_1,v_2}^{NN}(x) = (1 - \alpha)(2x)^{0.7} + \alpha M_{v_1,v_2}^{0.7} \quad (0 \leq x \leq M_{v_1,v_2}) \quad (B.1) \]

\[ f_{v,e}^{NE}(x) = 2(1 - \alpha)x + \alpha M_{v,e} \quad (0 \leq x \leq M_{v,e}) \quad (B.2) \]

\[ f_{G}^{EE}(x) = \frac{16}{\pi^2} - 4\alpha x^2 + \alpha \frac{\pi^2}{4} \quad (0 \leq x \leq \frac{\pi}{2}) \quad (B.3) \]
B.1.1 Minimum penalty fraction: $\alpha$

In Section 3.4.4, we describe the semantics of the minimum penalty fraction, $\alpha$, that appears in Equation B.4. The minimum penalty fraction is a trade-off between the ability to distinguish no overlap from touching overlap and the ability to distinguish the different amount of overlap.

$$(NN) \quad \beta = (1 - \frac{1}{5\pi})\alpha + \frac{1}{5\pi} \approx 1.625 - 0.625\alpha \quad (0 < \alpha < 1)$$

$$(NE) \quad \beta = 2 - \alpha \quad (0 < \alpha < 1)$$

$$(EE) \quad \beta = \frac{16}{\pi^2} - 3\alpha \quad (0 < \alpha < \frac{16 - 2\pi}{3\pi^2} \approx 0.328)$$

**Theoretical analysis**

We show the function plot for NN overlap between bigger nodes, as mentioned in Section 3.4.4.

Figure B.1: The NN penalty mapping function of a big metanode pair that is 10x the size of the smallest node ($M = 10$), with different minimum penalty fraction ($\alpha$) values.
Computational analysis

We validate the theoretical analysis and generate practical suggestions with our implementation and three groups of synthetic small graph layouts, namely, progression-NN, progression-NE, and progression-EE.

We tested the influence of $\alpha$ for NN, NE, and EE separately as it is not necessary to use the same $\alpha$ for all three cases. We used six different $\alpha$ values within its valid range defined in Equation B.4: a near-minimum, a near-maximum, and four evenly-spaced values in between.

The progression-NN has a series of graph layouts with minimum, some, and near-max node-node overlap for either leaf nodes only, metanodes only, or both, as shown in Figure B.2. For each $\alpha$, we can compare the area-aware metrics between different layouts to obtain a practical sense of whether the differences in penalties are too little, too much, or just enough to distinguish the layouts. The same analysis also applies to NE (Figure B.3) and EE (Figure B.4).

With the extremely small $\alpha$ value 0.01, the near-minimum penalty for the touching overlap between leaf nodes are only 0.5 (average penalty is $P/C = 0.50/4 = 0.125$), which is too little to distinguish it from no overlap at all. With $\alpha$ values that are larger than 0.8, the differences between penalties for touching, some, and near-max overlap of leaf nodes are too tiny ($3.29 \rightarrow 3.91 \rightarrow 4.24$ when $\alpha = 0.80$), and differences for metanodes are also too tiny ($12.40 \rightarrow 15.34$). Note that the last difference seems sufficient ($15.34 \rightarrow 27.24$) only because the last layout has overlaps of both leaf node pairs and metanode pairs. With $\alpha$ values in between, the differences between penalties are generally sufficient to tell these layouts apart.

The choice of $\alpha$ is subjective to user preference, and they can have different $\alpha$ values for NN, NE, and EE, as long as it is within the range specified by Equation B.4. For simplicity, we use $\alpha = 0.20$ for all three cases.
Figure B.2: Influence of minimum penalty fraction ($\alpha$) tested on a small 2-level synthetic layout with increasing node-node overlap of leaf node pairs and metanode pairs.

<table>
<thead>
<tr>
<th>Overlap amount</th>
<th>Near-min</th>
<th>Some</th>
<th>Near-max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf node pairs or metanode pairs</td>
<td>Leaf nodes</td>
<td>Metanodes</td>
<td>Leaf nodes</td>
</tr>
<tr>
<td>Count</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Area-aware metrics (variable: $\alpha$)</td>
<td>0.01</td>
<td>0.50</td>
<td>1.69</td>
</tr>
<tr>
<td></td>
<td>0.20</td>
<td>1.17</td>
<td>4.27</td>
</tr>
<tr>
<td></td>
<td>0.40</td>
<td>1.88</td>
<td>6.98</td>
</tr>
<tr>
<td></td>
<td>0.60</td>
<td>2.59</td>
<td>9.69</td>
</tr>
<tr>
<td></td>
<td>0.80</td>
<td>3.29</td>
<td>12.40</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>3.96</td>
<td>14.97</td>
</tr>
</tbody>
</table>

Figure B.3: Influence of minimum penalty fraction ($\alpha$) tested on a small 2-level synthetic layout with increasing node-edge overlap of leaf node-edge pairs and metanode-edge pairs.

<table>
<thead>
<tr>
<th>Overlap amount</th>
<th>Near-min</th>
<th>Some</th>
<th>Near-max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf node pairs or metanode pairs</td>
<td>Leaf nodes</td>
<td>Metanodes</td>
<td>Leaf nodes</td>
</tr>
<tr>
<td>Count</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Area-aware metrics (variable: $\alpha$)</td>
<td>0.01</td>
<td>3.38</td>
<td>12.15</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>3.50</td>
<td>12.11</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>3.62</td>
<td>12.07</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>3.75</td>
<td>12.03</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>3.87</td>
<td>11.99</td>
</tr>
<tr>
<td></td>
<td>0.99</td>
<td>3.99</td>
<td>11.96</td>
</tr>
</tbody>
</table>
Figure B.4: Influence of minimum penalty fraction ($\alpha$) tested on a small single-level synthetic layout with decreasing edge-edge crossing angles.
B.1.2 EE curve shape: $\gamma$

For the EE case, we compare the quadratic function (Equation B.3) against a linear function, stated as follows.

$$f_{EE}^G(x) = \left(\frac{4}{\pi} - 2\alpha\right)x + \alpha\frac{\pi}{2} \quad (0 \leq x \leq \frac{\pi}{2}), \quad \text{where} \quad 0 < \alpha < \frac{2}{\pi} \approx 0.637$$

(B.5)

**Theoretical analysis**

To separate the influence of $\alpha$ and $\gamma$, we compare the EE linear function with the quadratic one using the same $\alpha$, but repeat the comparison for multiple different $\alpha$ values, as shown in Figure B.5. Since the domain of $\alpha$ is different for linear function, $(0, 0.637)$, and quadratic function, $(0, 0.328)$, we only compare the functions with valid $\alpha$ values for both, i.e., 0.01, 0.1, 0.2, 0.3.

We observe that the basic shapes and relationships of the two functions stay the same across different $\alpha$ values, despite that the touching penalty and range of penalty vary (in the way we described in the previous section). The absolute difference between the two functions decreases as $\alpha$ increases, due to the decrease in the range of penalty. The relative difference between two functions becomes large only when the complementary angle $x$ is nearly maximum, $\pi/2$, i.e., the glancing angle; otherwise, they are very close to each other.

Therefore, we can choose either the linear or quadratic function depending on how heavy the glancing angle should be penalized.

**Computational analysis**

We computed the area-aware metrics using both linear and quadratic penalty mapping functions on all 52 layouts we collected. Figure B.6 shows the penalties com-
Figure B.5: Two different curve shapes (linear and quadratic) of the EE penalty mapping function.

Figure B.6: EE area-aware metric with linear penalty mapping function against that with the quadratic function, using log scale on both axes.

puted with the linear function (x-axis) compared to that with the quadratic function
(y-axis). Each dot represents a graph layout in our dataset. We used log scales for both axes to de-clutter the dots as there are many small synthetic layouts. We can clearly see that the two penalties are linearly correlated. The Pearson correlation between the two penalties is 0.9997, and the slope of the linear regression line is 1.17, which indicates that penalties with the quadratic function are slightly bigger. In general, the difference between the two functions are very small.

We also sorted the layouts by the difference between the penalties using the two functions, and manually inspected a few layouts with the biggest difference. We found out that in these layouts, the angles between crossing edges are smaller than those in other layouts, and there are many near-glancing angles. This discovery also confirms our theoretical analysis: difference only becomes relatively large on small crossing angles (i.e., large complementary angles).

We chose the quadratic function for our analysis as Huang et al. found that the quadratic trend is more relevant to user performance in their study [83].

### B.2 Pseudo-code for NE and EE area-aware metric
Algorithm 5: Computation of node-edge area-aware metrics.

Input : $G = (V, E)$
Output: total penalty $p_{NE}(G)$ and count $C_{NE}(G)$

1. totalPenalty ← 0
2. count ← 0
3. for $v \in V$ do
   4. for $e \in E$ do
      5. if $v \notin e.ends$ && !IsAncOrDesc($v$, e.ends) then
         6. if CheckIntersection($v$, e) then
            7. $x$ ← ComputeOverlapLength($v$, e)
            8. penalty ← PenaltyMapFunc($x$, $v$, e)
            9. totalPenalty ← totalPenalty + penalty
            10. count ← count + 1
   6. return totalPenalty, count

Algorithm 6: Computation of edge-edge area-aware metrics.

Input : $G = (V, E)$
Output: total penalty $p_{EE}(G)$ and count $C_{EE}(G)$

1. totalPenalty ← 0
2. count ← 0
3. for $e1 \in E$ do
   4. for $e2 \in E$ do
      5. if $e1 \neq e2$ then
         6. if CheckIntersection($v$, e) then
            7. $x$ ← ComputeCrossingAngle($e1$, $e2$)
            8. penalty ← PenaltyMapFunc($x$)
            9. totalPenalty ← totalPenalty + penalty
            10. count ← count + 1
   6. return totalPenalty, count
<table>
<thead>
<tr>
<th>Type</th>
<th>Name (source)</th>
<th>#Nodes</th>
<th>#Edge</th>
<th>#Layouts</th>
<th>Layout algorithm</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>four-clusters</td>
<td>40</td>
<td>123</td>
<td>10</td>
<td></td>
<td>debugging</td>
</tr>
<tr>
<td></td>
<td>proxy shape</td>
<td>19</td>
<td>40</td>
<td>2</td>
<td></td>
<td>exploration of proxy shapes</td>
</tr>
<tr>
<td></td>
<td>meta-edges 1</td>
<td>43</td>
<td>114</td>
<td>2</td>
<td></td>
<td>meta-edges overlap</td>
</tr>
<tr>
<td></td>
<td>meta-edges 2</td>
<td>42</td>
<td>82</td>
<td>1</td>
<td>Manual</td>
<td>meta-edge and node overlap</td>
</tr>
<tr>
<td></td>
<td>touching</td>
<td>12</td>
<td>18</td>
<td>2</td>
<td></td>
<td>minimum overlap</td>
</tr>
<tr>
<td></td>
<td>progression NN</td>
<td>40</td>
<td>123</td>
<td>7</td>
<td></td>
<td>analysis of minimum penalty fraction ((\alpha)) for NN</td>
</tr>
<tr>
<td></td>
<td>progression NE</td>
<td>40</td>
<td>123</td>
<td>6</td>
<td></td>
<td>analysis of minimum penalty fraction ((\alpha)) for NE</td>
</tr>
<tr>
<td></td>
<td>progression EE</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td></td>
<td>analysis of minimum penalty fraction ((\alpha)) for EE</td>
</tr>
<tr>
<td></td>
<td>single-level</td>
<td>31</td>
<td>92</td>
<td>4</td>
<td></td>
<td>check metrics performance on this widely-used type of graph</td>
</tr>
<tr>
<td></td>
<td>variable node-size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-world</td>
<td>coauthor-big [87]</td>
<td>1538</td>
<td>8040</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>coauthor-small [7]</td>
<td>103</td>
<td>505</td>
<td>4</td>
<td>GEM [52], FM3</td>
<td>various situation in large layout</td>
</tr>
<tr>
<td></td>
<td>email-eu-core [102]</td>
<td>986</td>
<td>15957</td>
<td>4</td>
<td>Grouseflocks [7], Koala [87]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>partition-add32 [160]</td>
<td>4960</td>
<td>9462</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>13</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table B.1:** Full list of graph layouts

### B.3 List of graph layouts

See Table B.1
B.4 Computational time

Table B.2 shows detailed running times on each layout for three metric families (NN, NE, and EE) and two approaches (AS vs. count), averaged over 4 runs. The slowdown factor is the ratio between running time of AS and count, representing how much the AS metric is slower than traditional count-based metrics. It is computed only if the running time is greater than 1 second, in order to reduce timing errors of the operating system. The average slowdown at the last row is computed over the valid rows.
<table>
<thead>
<tr>
<th>Layout</th>
<th>sprawler count</th>
<th>slowdown</th>
<th>sprawler NE count</th>
<th>slowdown</th>
<th>sprawler EE count</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>four-clusters-ne0</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.04</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-ne1</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.07</td>
<td>0.05</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-mil</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.04</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-mil1</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.10</td>
<td>0.10</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-mil2</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.09</td>
<td>0.09</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-mil3</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.07</td>
<td>0.07</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-mil4</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.08</td>
<td>0.08</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-sprawl</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.06</td>
<td>NA</td>
</tr>
<tr>
<td>four-clusters-sprawl1</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.11</td>
<td>0.10</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-1</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-2</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-3</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.08</td>
<td>0.08</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-4</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.10</td>
<td>0.10</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-5</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.05</td>
<td>0.05</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-6</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.09</td>
<td>0.09</td>
<td>NA</td>
</tr>
<tr>
<td>special-cases-7</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
</tr>
<tr>
<td>progression-in-none</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.06</td>
<td>NA</td>
</tr>
<tr>
<td>progression-nn-none</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.07</td>
<td>0.07</td>
<td>NA</td>
</tr>
<tr>
<td>progression-nn-touch-meta</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.07</td>
<td>0.07</td>
<td>NA</td>
</tr>
<tr>
<td>progression-nn-touch-leaf</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.10</td>
<td>0.10</td>
<td>NA</td>
</tr>
<tr>
<td>progression-nn-some-meta</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.09</td>
<td>0.09</td>
<td>NA</td>
</tr>
<tr>
<td>progression-nn-some-leaf</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.08</td>
<td>0.08</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-touch-meta</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.09</td>
<td>0.09</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-touch-leaf</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.06</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-some-meta</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.06</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-some-leaf</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.07</td>
<td>0.07</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-near-max-meta</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.07</td>
<td>0.07</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-near-max-leaf</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.06</td>
<td>0.06</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-ortho</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-half</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
</tr>
<tr>
<td>progression-ne-glancing</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
<td>0.00</td>
<td>0.00</td>
<td>NA</td>
</tr>
<tr>
<td>varied-davidson-harel</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.04</td>
<td>0.04</td>
<td>NA</td>
</tr>
<tr>
<td>varied-fast-multipole-emb</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.04</td>
<td>0.04</td>
<td>NA</td>
</tr>
<tr>
<td>varied-stress-majorization</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.03</td>
<td>0.03</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-grouseflocks-0</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.23</td>
<td>0.23</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-grouseflocks-1</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.28</td>
<td>0.28</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-grouseflocks-2</td>
<td>0.02</td>
<td>0.02</td>
<td>NA</td>
<td>0.94</td>
<td>0.94</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-grouseflocks-3</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.28</td>
<td>0.28</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-grouseflocks-4</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.94</td>
<td>0.94</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-grouseflocks-5</td>
<td>0.01</td>
<td>0.01</td>
<td>NA</td>
<td>0.28</td>
<td>0.28</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-koala</td>
<td>0.05</td>
<td>0.04</td>
<td>NA</td>
<td>0.63</td>
<td>0.63</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-gem</td>
<td>0.05</td>
<td>0.04</td>
<td>NA</td>
<td>1.09</td>
<td>1.09</td>
<td>NA</td>
</tr>
<tr>
<td>coauthor-koala</td>
<td>13.83</td>
<td>9.46</td>
<td>1.46</td>
<td>139.26</td>
<td>97.92</td>
<td>1.42</td>
</tr>
<tr>
<td>Average</td>
<td>1.38</td>
<td>1.50</td>
<td>1.85</td>
<td>239</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table B.2: Computation time of each layout for three metric families and two approaches.
B.5 Computational pipeline

The full computational pipeline that we have implemented includes multiple data preparation steps in addition to the sprawlter metrics themselves, plus other metrics for comparative analysis purposes.

1. **Convert graph format** (Python). We convert a variety of input graph formats into Tulip format, as Tulip [10] provides several useful layout algorithms.

2. **Apply layout algorithms.** For the synthetic graphs, we manually position the nodes to create various degree of clutter and sprawl, and also apply four different layout algorithms within the version 5.2.1 of Tulip [10] (fast multipole embedder, Davidson-Harel, Linlog, stress majorization). For the real-world graphs, we apply two layout algorithms in Tulip (GEM, FM3); we use the 2008 version of GrouseFlocks [7], and a version of Koala [87] modified to save in Tulip format. The output in all cases is the geometric information of nodes and edges (position, area, shape, length) in Tulip format.

3. **Extract geometries and node hierarchy** (Python). We then parse the 56 layouts in Tulip format, extract the information of geometry and node hierarchy, and store it in JSON format.

4. **Compute metrics** (Python). We implement the sprawlter metrics for the NN (algorithm 4), NE, and EE families and also their respective count-based metrics, and also the global readability metrics of node-node overlap and crossing angle of Dunne et al [39]. We use the Shapely Python package [157] for manipulation of geometry.

5. **Display and analyze results for comparison.** We display the computed metrics of each layout in a table with HTML and JavaScript, and we write Python scripts to analyze parameters and computational time.

GrouseFlocks was developed over ten years ago, and while it still functions as an interactive exploration tool for multi-level graph layouts, the coloring of metanodes
is no longer correct (possibly due to deprecated dependent libraries). In order to have easy-to-understand results figures, we exported Tulip-format output files from GrouseFlocks into the current Tulip, saved them in SVG format, and placed the metanodes with the correct colors back to their original positions in Adobe Illustrator.

In the supplemental materials, we include all input files (in JSON format after step 3), output files (results of step 4), implementation code, and Python scripts for results analysis.
Appendix C

Supplemental Material for LogSeg

C.1 Features

C.1.1 Commands

We exported 100 million events which contained 1949 unique commands from the database in May 2018. The command frequencies follow a long-tail distribution: a few commands are frequently used, whereas many commands are rarely used. We inspected the distribution of commands ordered by frequencies as a data quality check, particularly those at the head of the distribution. Examples of the most frequent commands are Open, Move, and Crop. The actual number of unique commands in the database is larger than the number of menu items in the PS interface (about 1400), due to non-standard commands like Photoshop (PS) Actions\(^1\) and multilingual versions of the same command. Many of the infrequent ones are

\(^1\)Actions and the Action Panel in Photoshop: [https://helpx.adobe.com/photoshop/using/actions-actions-panel.html](https://helpx.adobe.com/photoshop/using/actions-actions-panel.html)
the user-defined Actions that are personal rather than universal and thus are not useful for us.

We also note that the command name alone does not always contain enough semantic information to fully disambiguate user intention or the task at hand, however. For example, Brush Tool, one of the most frequently used commands, can be used to either paint colors on canvas or edit a layer mask.

Grouping these events by session identifier and ordering them by timestamp, we reconstructed 169,387 sessions. We specifically define a session to be a temporal sequence of events triggered by a user when completing a PS task.

We used word2vec [122] to learn a vector for each command. We conducted a sanity check of the embedding space using the Google Embedding Projector [158] with t-SNE [172] to inspect some of the clusters in the interface, and found that the structure of this space was indeed plausible. For example, in Figure C.1 the red circle contains many commands for changing layer blending options.

Figure C.2 shows the neighborhood around the Brush Tool command, which also has plausible command names such as Eyedropper Tool and Mixer Brush Tool.

Finally, Figure C.3 shows the neighborhood of the New Color Fill Layer command, which creates a new adjustment layer. Again, nearby commands such as Modify Color Fill Layer and Modify Levels Layer seem to capture reasonable similarity.
Figure C.1: Embedding space neighborhood of Linear Burn command.
**Figure C.2:** Embedding space neighborhood of Brush Tool command.
Figure C.3: Embedding space neighborhood of New Color Fill Layer command.
C.1.2 Layer

Layer hierarchy examples from three sessions (S1, S2, S14) are shown in Figure C.4. The annotations on the left show examples of layer relationships and the corresponding layer similarities.

The layer similarity feature computation requires complete information about the
layer hierarchy as it has evolved up to any specific time point, so it is derived using information spread across multiple individual event attributes.

The overall equation for layer similarity between two layers $A$ and $B$: If $A$ and $B$ are identical layers, the similarity is maximum (1.0); otherwise, it is the sum of layer similarity for duplicate layers, adjustment layers, and layer group, capped by 1.0. Note that in these equations, $A$ and $B$ are commutative.

$$S(A,B) = \begin{cases} 
1.0 & \text{if } A = B \\
\min(1.0, S_{dup}(A,B) + S_{adj}(A,B) + S_{grp}(A,B)) & \text{if } A \neq B 
\end{cases} \quad (C.1)$$

The similarity for duplicate layer is computed as follows.

$$S_{dup}(A,B) = \begin{cases} 
0.8 & \text{if } A \text{ duplicates } B \\
0 & \text{otherwise} 
\end{cases} \quad (C.2)$$

The similarity for adjustment layer is computed as follows.

$$S_{adj}(A,B) = \begin{cases} 
0.5 & \text{if } A \text{ is adjustment layer of } B \\
0 & \text{otherwise} 
\end{cases} \quad (C.3)$$

The similarity for layer group is computed as follows. The length of path between $A$ and $B$ in the layer hierarchy is denoted as $d$. Most cases are direct siblings ($d = 2$), and the layer similarity would be 0.5.

$$S_{grp}(A,B) = \begin{cases} 
\frac{1}{2^{d-1}} & \text{if } A \text{ and } B \text{ are in the same layer group} \\
0 & \text{otherwise} 
\end{cases} \quad (C.4)$$

### C.2 Data Collection
<table>
<thead>
<tr>
<th>Session ID</th>
<th>Participant ID</th>
<th>Task Type</th>
<th>Reference Image</th>
<th>#Events</th>
<th>Usage in model</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>Poster creation</td>
<td>![Image]</td>
<td>275</td>
<td>Training and validation set</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>Poster creation</td>
<td>![Image]</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>P3</td>
<td>Portrait retouching</td>
<td>![Image]</td>
<td>648</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>P4</td>
<td>Special effect creation</td>
<td>![Image]</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>P5</td>
<td>Portrait retouching</td>
<td>Same as S3</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>P5</td>
<td>Portrait retouching</td>
<td>![Image]</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>P6</td>
<td>Poster creation</td>
<td>![Image]</td>
<td>1064</td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td>P7</td>
<td>Poster creation</td>
<td>Same as S1</td>
<td>179</td>
<td></td>
</tr>
</tbody>
</table>

**Figure C.5:** 8 collected sessions in the first round.
<table>
<thead>
<tr>
<th>Session ID</th>
<th>Participant ID</th>
<th>Task Type</th>
<th>Reference Image</th>
<th>#Events</th>
<th>Usage in model</th>
</tr>
</thead>
<tbody>
<tr>
<td>S9</td>
<td>P8</td>
<td>Portrait retouching</td>
<td><img src="image1" alt="Image" /></td>
<td>428</td>
<td>Training and validation set</td>
</tr>
<tr>
<td>S10</td>
<td>P8</td>
<td>Portrait retouching</td>
<td><img src="image2" alt="Image" /></td>
<td>180</td>
<td>Test set</td>
</tr>
<tr>
<td>S11</td>
<td>P9</td>
<td>Poster creation</td>
<td><img src="image3" alt="Image" /></td>
<td>261</td>
<td>Training and validation set</td>
</tr>
<tr>
<td>S12</td>
<td>P10</td>
<td>Poster creation</td>
<td><img src="image4" alt="Image" /></td>
<td>323</td>
<td>Test set</td>
</tr>
<tr>
<td>S13</td>
<td>P11</td>
<td>Special effect creation</td>
<td><img src="image5" alt="Image" /></td>
<td>542</td>
<td></td>
</tr>
<tr>
<td>S14</td>
<td>P5</td>
<td>Portrait retouching</td>
<td>Same as S9</td>
<td>426</td>
<td></td>
</tr>
<tr>
<td>S15</td>
<td>P12</td>
<td>Poster creation</td>
<td>Same as S11</td>
<td>380</td>
<td></td>
</tr>
<tr>
<td>S16</td>
<td>P13</td>
<td>Poster creation</td>
<td>Same as S12</td>
<td>297</td>
<td>Training and validation set</td>
</tr>
</tbody>
</table>

**Figure C.6:** 8 collected sessions in the second round.
Table C.1: Summary of task type

<table>
<thead>
<tr>
<th>Task type</th>
<th>#Sessions in 1st round</th>
<th>#Sessions in 2nd round</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poster creation</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Portrait retouching</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Special effect creation</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

251
C.3 Hyperparameter Tuning for SVM

We briefly review how a trained SVM model predicts a label given an event. It uses a kernel function that takes the feature vector as input and gives signed distance from the data point to a separating hyperplane as output, which is then converted to the probability of true prediction. Finally, it compares this probability score to a threshold \( t \) (model hyperparameter): if probability is greater than \( t \), then this event is predicted as boundary.

There are two hyperparameters in our model: window size \( k \), the number of previous events used in the feature vector, and threshold \( t \), to determine the probability value where an event is predicted to be the boundary. We train the model on the training set, and measure performance on the validation set to tune these hyperparameters, following common practice in machine learning.

C.3.1 Window size \( k \)

For each window size \( k \) from 1 to 10, we assess model performance after model training by computing and plotting its receiver operating characteristic (ROC) curve, shown in Figure C.7. The ROC curve is drawn by varying the hyperparameter threshold \( t \) from 0 to 1 and plotting a point on the curve for each value of \( t \). The x axis is the false positive rate, and the y axis is the true positive rate for that threshold value. The area under curve (AUC) of a ROC curve indicates the probability that the model will rank a randomly chosen boundary event higher than a randomly chosen non-boundary event. A common interpretation is that the larger the area is, the better performance the model has. From qualitative inspection of Figure C.7, we can see that the 10 curves are closely intertwined with each other, and they are all far above the diagonal curve representing random guesses, indicating that they are similarly good classifiers. Quantitatively, we can see that the AUCs are also highly similar (min: 0.938, max: 0.962).

We conclude that window size does not influence model performance, and thus we choose \( k = 1 \) to reduce the size of feature vectors.
C.3.2 Threshold $t$

After choosing the window size, we investigate the performance of the model with $k=1$ under different threshold values. We plot precision (red line) and recall (blue line) against threshold $t$ (x-axis) in Figure C.8. The trade-off between precision and recall is easy to spot: as threshold increases, precision increases but recall drops. Considering the downstream applications of the segmentation model, we favor recall over precision. That is, we are willing to trade precision for higher recall because there are many fewer positive labels (boundaries) than negative labels (non-boundaries) in the dataset. It is important to correctly identify as many boundaries as possible, and false negatives (boundaries predicted as non-boundaries) are undesirable as it is not easy for end users to identify these missed boundaries. On the other hand, false positives (predicting non-boundaries as boundaries) are less detrimental: although these errors lead to over-segmentation of the logs, users can
quickly dismiss them through an interactive interface. We use the F2 score (thick black line), which weighs recall twice as much as precision, to quantify this trade-off choice.

From Figure C.8, we find a clear winner, \( t = 0.24 \), which yields the highest F2 score (0.74), as the threshold for our lowest-level segmentation. The corresponding recall is 0.84, and precision is 0.50.
C.4 Results

C.4.1 Labeling interface

Figure C.9 shows the interface used by authors of the CHI paper (Zipeng Liu and Zhicheng Liu) when labeling the sessions. It shows the logged attributes (layer, command, image content, duration) and a few derived attributes such as the diff image.
### Logs

<table>
<thead>
<tr>
<th>Thumbnail</th>
<th>DiffImage</th>
<th>sequenceId</th>
<th>DiffScore</th>
<th>Overlap</th>
<th>duration</th>
<th>ElapsedTime</th>
<th>EventName</th>
<th>ActiveLayer</th>
<th>ActiveLayerName</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>13.448</td>
<td>Place Embedded Smart Object</td>
<td>8</td>
<td>S040b9645683053d572a60cE3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>3.206</td>
<td>Add Layer Mask</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>2.51</td>
<td>Quick Selection</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>5.121</td>
<td>Delete Layer Mask</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>5.774</td>
<td>Add Vector Mask</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>1.073</td>
<td>Quick Selection</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>5.992</td>
<td>Delete Layer Mask</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>6.172</td>
<td>Add Layer Mask</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>26.924</td>
<td>Quick Selection</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>4.856</td>
<td>Stop Backward</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>21.587</td>
<td>Add Layer Mask</td>
<td>9</td>
<td>c99f-18666032_5920</td>
</tr>
</tbody>
</table>

**Figure C.9:** Labeling interface on a scene creation session (S1).
C.4.2 Visualization of sessions in test set

In Section 4.8 we show and inspect 2 out of the 5 sessions (S10, S15) in the test set, and here we show the other 3 test sessions (S12, S13, S14). We include each of the 16 sessions as high-resolution individual PNG files in supplemental materials as well.

Figure C.10 and Figure C.11 show the visualization of session S12, for a poster creation task. The participant works in a highly organized way, which results in many levels in the human labels. Most of the boundaries in human labels (gaps between grey chunks) for both low- and high-level are also predicted as boundaries in red and blue.

There are a few low-level missing boundaries. For example, in the top highlighted box, participant moved a sphere icon (there are 3 icons in total) using only one event, and then he continued adjusting the positions of the three icons. The model fails to detect this boundary as it thinks the participant is still working on the icons in the icon layer group (high layer similarity) with highly similar commands (Move, FreeTransform).

There are many over-segmentations. In the bottom highlight box, the participant tries to put a few layers into a layer group, but they were having trouble locating them in the layer panel, resulting in a series of events on unrelated different layers until he found the correct ones. In this case, the user is having trouble keeping track of the interface, so it would naturally results less structure in the logged data.
Figure C.10: Analysis of segmentation results for session S12 (upper part).
Figure C.11: Analysis of segmentation results for session S12 (lower part).
Figure C.12 and Figure C.13 show session S13, a special effect task to turn a day image of a city street into night. The computational results align with the human labels very well for this session, but it comes with a lot of over-segmentation. In the highlighted box, the participant frequently copies and hides a layer as its backup as a way of smart version control: marking the milestone states such that they could go back at anytime later. It involves a lot of layer switching and toggling, leading to many single-event low-level chunks in the computed results.

However, in the high-level chunks (red), the model successfully removes the over-segmentation while keeping the true boundaries in the human labels.
Figure C.12: Analysis of segmentation results for session S13 (upper part).
Figure C.13: Analysis of segmentation results for session S13 (lower part).
Figure C.14 and Figure C.15 show session S14, a portrait retouching task to restore an old photo. For this session, there are noticeably less chunks in both computed results and human labels. It is also noticeable that the precision is high and recall is low (compared to other sessions). The many missing boundaries (highlighted) happen during the first half of the session, where the participants were fixing cracks on different areas on the photo (upper part, head and face, bottom left, back to face, and clothes) but on the same layer. Each event only fixes a small amount of pixels, which leads to the model’s failure to capture the change of semantic areas of the photo. In the second half of the session, where the participants were working on the eyes using multiple copy-and-paste phases, the model is able to find true structures that aligned with the human labels.
Figure C.14: Analysis of segmentation results for session S14 (upper part).
Figure C.15: Analysis of segmentation results for session S14 (lower part).
C.5 Rule-based model

In the early phase of the project, we considered a few rule-based models before coming up with the two-stage model described in Section 4.6. We present one of them here.

In the data collection sessions with Photoshop experts, we observed that when they switched to a new layer or use a different command, it is likely that they finished a subtask and started the next one, especially in poster creation tasks. Therefore, we created a model that segment the event sequence whenever there is a layer switch or command switch. Figure C.16 illustrates the model for session S1, a poster creation session. Note that the visualization is different from the previous figures in this document as they were created in the early phase of this project. Each row represents a layer in the session, and the rectangular blocks represent logged events, sorted by timestamps from left to right. The blue triangles are the ground-truth labels, while the red vertical lines are the predicted boundaries from the rule-based model. We also shows a snapshot image of the current canvas when a “difference score” (computed with layer, command, and image difference) is high from the previous event. We can see that there are many cases of over-segmentation (false positives) and missing boundaries (false negatives). Figure C.17 shows part of a portrait creation session, where the user tends to create less layers and more likely to work on the same layer even if (s)he is doing a different subtask. The errors of the rule-based models are higher in such tasks.
Figure C.16: Illustration of a rule-based model on S1, a poster creation session. The model segments at any events with a layer or command switch. Events are sorted by time and shown from left to right. The top screenshot is the left side of the whole image, the bottom one is the right side.

Figure C.17: Partial illustration of rule-based model on S3, a portrait retouching session.
Appendix D

Supplemental Material for CorGIE

D.1 Concrete task examples

As part of our iterative refinement of the task abstraction, we explicitly analyzed examples of concrete tasks with two datasets. The first restaurant-user dataset is from Yelp \(^1\). The second dataset is the Cora academic papers citations graph (second motivating scenario, described in Section 5.2.2). We structured the abstracted versions of these concrete tasks according to the nature and number of the targets. The cluster-oriented targets are one cluster, two clusters directly compared, multiple clusters, and outliers versus existing clusters. The node-oriented targets are one node vs. all others, two nodes directly compared, and three nodes directly compared. This analysis took place during an intermediate stage, before we separated graph space into topology space vs. feature space in the data abstraction. It was a stepping stone towards our simpler high-level task abstraction that treats nodes and clusters in a more unified way, and allowed us to cross-check that the CorGIE design would support many of these lower-level tasks.

\(^1\)https://www.yelp.com/dataset
D.2 Configurable settings, filters and menus

The visual appearance of nodes is user-configurable in the SETTINGS VIEW. Nodes can be colored by their UMAP position (i.e. the rainbow color scheme), labeling and node type, as shown in Figure D.2. To highlight nodes with specific properties, we allow users to brush the nodes of a specific node type or node label, as shown in Figure D.3. Users can also search a specific node by its label or ID, as shown in Figure D.4.

D.3 Alternative visual design for topology space

In Section 5.4.2, we visualize the topology with node-link diagrams: an initial layout for a static global overview, and a focal layout for hops of neighbors of focal nodes and their clustering structure (Section 5.5). Before finalizing this visual design, we tried several alternatives which turned out not suitable for our visualization tasks.
**Figure D.2:** Choices of node colors in the (a) Movie dataset and (b) Cora dataset.

**Figure D.3:** Highlight nodes with specific properties in the (a) Movie dataset and (b) Cora dataset.
Figure D.4: Search a node by its label in the Movie dataset.

Figure D.5: Bundled edges can be shown in the latent space. This demonstrates the bipartite graph in the Movie dataset, with movie nodes in blue and user nodes in orange.
The high-level goal of topology space views is to show shared neighbors of the user-specified focal nodes. As we have derived a neighbor set for each node, we considered techniques for set visualization. We thought that the UpSet approach [104] could be suitable for our problem, since it could show the intersections of combinations of multiple sets, specifically the neighbor set intersections of the focal nodes.

Following this idea, we designed a UpSet style visualization, as shown in Figure D.7. Each row represents the intersection of a specific combination of sets, e.g., intersection of the neighbor set of Avatar and that of Fight Club. The circular dots on the left represent which neighbor sets are in this combination. The columns on the right, with each for a node type, represent the cardinality of the intersection. For example, with four focal movies, the 8th row in Figure D.7 tells us that movie 2 and 3 have 1 cast member in common, 1 production company in common, and 4 genres in common.

It is not enough to show the intersection cardinalities: users also need to know the identities of the shared neighbors, that is, the intersecting nodes. To visualize the identities of neighbors, we chose a matrix representation (Figure D.8), where the rows are for focal nodes and columns are for their neighbors. It can be viewed as
Figure D.7: UpSet style visualization for neighbor sets of four focal movie nodes. The input graph has five node types: movie, cast, production company, genre, and user.

A partial adjacency matrix, with the neighbors ordered by their popularity in the focal nodes. In addition to the matrix, there is a histogram at the bottom to show the popularity count number. As there could be many neighbors, we compress the matrix columns into an intersection overview (Figure D.9) using the popularity. At this point, rather than pursuing this line of attack, we moved on to other approaches.

There are a few drawbacks of these three views. First, they suffer from visual scalability: the Upset style visualization has a power set scaling factor in the number of rows; the partial adjacency matrix has many columns; the intersection overview can potentially have many rows. Second, they are too sensitive to absolute node counts or node order, which might lose the context. Third, it is difficult to extend them to compare two sets of focal nodes.
Figure D.8: Partial adjacency matrix, alternative visual design for showing neighbor set intersections of focal nodes.

Figure D.9: Intersection overview for showing counts of neighbor set intersections of focal nodes. This view compresses the columns in Figure D.8. The numbers within matrix cells could of course be visually encoded, but instead we moved on to other design approaches.
D.4 Results

In Chapter 5, we present two usage scenarios and the session with one of the two expert P1. We systematically describe four aspects of CorGIE usage: the visualization task being conducted, the visual operation used within the CorGIE interface, the direct observation that is possible from the CorGIE display, and any resulting inference of authors or participants. In Section 5.6, we only color-code the visualization task with color, and here we provide a version with all four aspects color-coded.

We also describe the other expert session with P2 in Section D.4.4.

D.4.1 Usage scenario 1: Movie Recommendation

To obtain an overview of the dataset, we first color the nodes by their node type (user and movie). Figure D.11a shows that we can see two clusters of movies in blue on the right-hand side and many clusters of users in orange in the rest of the latent space. After coloring the nodes by the underlying positional colormap in the latent space, we observe in the GLOBAL TOPOLOGY view (Figure D.10g) that most user nodes (triangles) are laid out around movie nodes (circles). Many nodes that are connected by only one edge are placed around the periphery of the layout, indicating that there are many users who only watched one movie. In Figure D.11b, we notice that the peripheral users that are connected to the same movies have similar colors: for example, the triangles on the bottom right are all green. We can thus infer that the GNN does a good job in grouping these one-time users reviewing the same movie.

To compare user clusters, we select and focus on two clusters from the LATENT SPACE, one in the green zone (foc-0) on the left, the other in the purple zone (foc-1) on the upper right, as shown in Figure D.10. In the K-HOP TOPOLOGY (Figure D.10b), there appears to be three green sub-clusters in foc-0, and the green nodes connect to many nodes in the hop-1 box, while foc-1 does not show any salient internal structure. To understand the topological difference between the
Figure D.10: Full screenshot of CorGIE interface on the Movie dataset, with two focal groups of user nodes. The views, with names shown in grey boxes, are laid out in four major areas on the screen, shown in blue: (a) the LATENT SPACE VIEW is for 2D node positions in latent space; (c) the NODE FEATURES VIEW is for feature distribution of all and focal nodes; (f & g) the TOPOLOGY VIEWS show local topological neighbors and global topology; (b) the LATENT NEIGHBOR BLOCKS VIEW and the (e) DISTANCE COMPARISON VIEW connect different spaces. The fifth area is for toggles and menus: (d) the SETTINGS VIEW.

Besides graph topologies, we also want to check whether distances match between spaces. In the DISTANCE COMPARISON VIEW (Figure D.11e), we can see that the within group distances in latent space are small, while the between group distances are large, which confirms that we have picked two distant clusters from the LATENT SPACE. We notice that the topological distances within the green
Figure D.11: Check overview of the Movie scenario: (a) LATENT SPACE, movie nodes in blue and user nodes in orange; (b) inset of GLOBAL TOPOLOGY showing similar peripheral nodes in similar green color (closeup of lower right for screenshot in Figure D.10). When comparing two clusters of user nodes, (c) we select nodes in foc-1 of the K-HOP TOPOLOGY, highlighting themselves and four neighbors in the hop-1 box, and also in the (d) GLOBAL TOPOLOGY. (e) We compare distances between topology and latent space, and (f) between feature and latent space.

cluster (foc-0) are relatively large considering they belong to the same cluster (left-most vertical histogram in Figure D.11e), which actually matches the existence of three sub-clusters in the K-HOP TOPOLOGY. The feature distances in all three charts of Figure D.11f are small and similar, indicating that the node features
Figure D.12: Check movie recommendations: (a) top 5 recommended movies for user 5355; (b) K-HOP TOPOLOGY focusing user 5355 and movie \textit{LotR}; (c) hover on the movie \textit{Suicide Squad} that the user has watched before; (d) topological distance between 5355 and \textit{LotR} is large (0.97); (e) K-HOP TOPOLOGY for user 587 and movie \textit{LotR}.

(\#votes and average vote of a user) cannot distinguish \textit{foc-0} and \textit{foc-1}. We further confirm the ineffectiveness of the features by reading the feature distances by picking some other node groups (not shown in figure), which indicates that they are not useful features, and could perhaps be removed from the dataset.

After exploring the clustering structure, we inspect instances of recommendation. In Figure D.12a we select the user node 5355, and list its top 5 recommended movies. The first one is \textit{The Lord of the Rings: the Return of King (LotR)}. We would like to understand why GNN decides to recommend \textit{LotR}, so we focus this recommendation by clicking on it. CorGIE automatically creates two focal groups with the user 5355 in \textit{foc-0} and movie \textit{LotR} in \textit{foc-1} (Figure D.12b). The K-HOP TOPOLOGY shows that user 5355 watched two movies (in pink) before. When we hover on the two pink nodes in the hop-1 box (Figure D.12c), we find out that each only shares a few users with the recommended movie \textit{LotR}; for example, \textit{Suicide Squad} has only 5 hop-1 shared neighbors. We believe that this recommendation is poor, potentially indicating that we are not there yet with GNN training. We further confirmed this problem by reading the topological 0.05 and latent 0.97 distances between user 5355 and movie \textit{LotR} (Figure D.12e), which seem to be correlated negatively.

We later find a recommendation that makes sense: user 587 and movie \textit{LotR}. As shown in Figure D.12c, the two movies that 587 watched, \textit{Inception} and \textit{The Dark Night}, share many users with the recommended movie \textit{LotR} (many hop-1 neigh-
bors are highlighted when hovering on *Inception*). Moreover, the topological distance is 0.72 in this case, which is relatively small compared to the overall distance distribution (not shown in figure). CorGIE thus shows evidence that this recommendation is well supported.

We repeat this process to check other recommendations, and we find many that do not make sense. We conclude that this training result is not satisfactory. It could be due to the “cold start” problem in such a small dataset, where most users only watch 1 or 2 movies. Also, the node features were not very useful: e.g., we know that #votes cannot distinguish users effectively. To improve the recommendation, we might try a different model or fix the dataset problems.
D.4.2 Usage scenario 2: Cora

As with the *Movie* scenario, we first check the clustering structure of paper nodes. We color the nodes by the predicted labels to see if the label distribution makes sense. In **LATENT SPACE** (Figure D.13a), we can see that the different classes of papers are roughly separated to different areas. In **GLOBAL TOPOLOGY** (Figure D.13b), although the force-directed layout lacks much structure, we can still see that nearby nodes are in similar colors. In **LATENT NEIGHBOR BLOCKS** (Figure D.13c), we can see that the red-outlined origin and its surrounding cells are darker for all the blocks. All three observations indicate that the GNN has done a good job in classifying papers using the graph topology.

Next, we inspect and compare a few clusters. For example, when we compare (select and focus) the entire red cluster and a left part of the blue one (Figure D.13d), the **K-HOP TOPOLOGY** (Figure D.13e) shows that most nodes of the same color connect to each other, and the diff chart in the **NODE FEATURES VIEW** (Figure D.13f) signals a considerable amount of different words between the two paper clusters (many visible dark strips on the diff row). These observations reinforce our good impression on the training result.

As the GNN seems to classify most nodes successfully, we look for problematic nodes and edges. We brush the distance scatterplot to highlight the node pairs with large latent distances but small topological distances (i.e. the bottom right area), as shown in Figure D.14a, and we focus one of these problematic node pairs. In the **K-HOP TOPOLOGY** (Figure D.14b), we find through a few rounds of interactive hover that the two focal nodes only share one 1st-hop neighbor (circled in red), which has many nodes connected in the second hop. As the Jaccard distance accounts for multiple hops of neighbors, the large number of shared hop-2 neighbors can explain the low topological distance (0.23). We conjecture that the GNN decides to locate them far from each other due to the large difference in the first hop. Further investigation showed that the node in *foc-0* is mis-classified, which hints at the limitations of this GNN model to deal with such situations.
Figure D.13: Explore overview of Cora dataset, with nodes colored by predicted label: (a) LATENT SPACE; (b) GLOBAL TOPOLOGY; (c) LATENT NEIGHBOR BLOCKS with zoomed-in inset showing that the red-outlined cells are darker than others. Compare two paper clusters: (d) focus on the red and blue clusters in the LATENT SPACE; (e) K-HOP TOPOLOGY; (f) NODE FEATURES VIEW with inset showing the word count differences between foc-0 and foc-1.
Figure D.14: Find problematic node pairs in the Cora dataset: (a) select node pairs with high latent distance but low topological distance; (b) K-HOP TOPOLOGY focusing on a buggy node pair, with only one shared hop-1 neighbor (circled) and many shared hop-2 neighbors.
Figure D.15: P1 explored the decision boundary in the Cora dataset: (a) he focused on the wrongly predicted nodes within a cluster (in the bounding box) in the LATENT SPACE; he compared (b) feature distances of the entire graph to (c) those within the focused nodes; he inspected neighbors in the K-HOP TOPOLOGY that are colored by (d) predicted label and (e) ground-truth label.

D.4.3 Expert session P1: Cora decision boundary

In this case study, expert P1 first chose to perform quite similar tasks as in our usage scenario. Then he wanted to check the overview to study misclassification. He colored the nodes by label correctness (Figure D.15a), where he saw that the red (misclassified) nodes are distributed across different clusters. He used a filtered brush selection to highlight and focus the wrongly predicted nodes within the middle cluster in the LATENT SPACE. He colored the nodes by the predicted labels (Figure D.15d) and true labels (Figure D.15e) respectively, so he could understand which classes are wrong. It appeared that the GNN predicted these nodes as the orange class similar to their hop-1 neighbors, but the ground truth labels state that they belong to multiple different classes (blue, green,
red, etc) although the hop-1 neighbors are still orange. Further inspection with hover and brush selection shows that the focal nodes are loosely connected to their hop-1 neighbors, but the hop-1 neighbors are tightly connected to the hop-2 ones. Based on the variance in true labels and the sparsity in topology, P1 inferred that these nodes were sitting at the decision boundary between two or multiple classes. It was his first time to visually inspect the boundary nodes even though he had used this dataset for years. He emphasized the importance of understanding the decision boundary if he needed to improve the GNN model.

After exploring the topology, P1 turned to DISTANCE COMPARISON for more information on the node features, where he found the feature distances within the focal nodes are relatively large compared to the overall distribution (Figure D.15c&b). He conjectured that the current model performed as he expected, and it would be unlikely to classify the boundary nodes correctly even if he kept training more epochs. He also conjectured that the GNN should use the node features in a better way to correct these nodes. As he confirmed that the GNN has done a good job for most nodes except some boundary ones, he concluded that there is no need to keep training more epochs, and training should be stopped to avoid over-fitting.
D.4.4 Expert session P2: USAir

We present how P2 used CorGIE on the dataset USAir from his previous GNN paper. It is a homogeneous graph showing the air routes (2126 edges) between airports in the US (332 nodes). There are no node features in this graph. To obtain a node embedding, we used a GCN to predict links between the airports in an unsupervised setting.

After CorGIE loads (Figure D.16), P2 first checks the overview of the graph. The nodes are colored by their 2D latent space position. The latent space view shows that there are many clusters, and the one in the lower left (green) corner appears to be very different from the others. The initial global layout shows that most of the nodes connect to neighbors with the same colors, and the latent neighbor blocks view shows that the topological neighbors are placed nearby (the dark blocks are close to the red outlined block). We can infer that the GNN node embedding makes sense at a high level.

P2 proceeds to find buggy link predictions by the GNN. With the help of the author, he creates a filter to single out the node pairs that are connected in the input...
Figure D.17: Find buggy node pairs. (left) Popup customization dialog to pick the node pairs that are connected in the input graph but predicted as false by the GNN; (middle) the resulting 28 erroneous node pairs shown in the scatterplot of Distance Comparison View; (right) the 28 error node pairs in the list on the right side of that view.

graph but are predicted false by the GNN, as shown in Figure D.17, with the popup window that appears after clicking on the wrench icon in the upper right corner of the Distance Comparison View. He then clicks the first buggy node pair 124-283 in the list on the right side of that view, which automatically creates two new focus groups of a single node each and a new k-hop focal layout with those groups. He hovers on the focal nodes in the focal layout to check their neighbors (Figure D.18), and he finds out that they actually share many neighbors in both the first hop and the second hop. The topological distance, as shown in the distance comparison, is 0.19, which is a very low value, which further confirms what he finds in the focal layout. However, the latent distance is 0.50, which equals the threshold in GNN that separates true and false predictions. He infers that this erroneous predicted pair is located in the decision boundary area, making it difficult to predict.
Figure D.18: P2 focuses on the first buggy node pair (124-283), and hovers on one of the focal nodes to see its neighbors.
D.5 Semi-structured interview in user study

In the user study with GNN experts, we introduced and demonstrated CorGIE for about 40 minutes, and then they used CorGIE on their own datasets for about 30 minutes. Finally we conducted a semi-structured interview to gather their feedback. Here are the questions we used in the interview:

- What is your general impression of the tool?
- What are the most useful features of the tool? Please explain why?
- What are features that are less / not useful for you?
- What features are missing from CorGIE but will be very helpful?
- Are the visual encodings easy to understand?
- How do you like the interactions? Do they allow you to do what you want?
- Would you like to use the tool in your daily workflow (if there is only minimum pre-processing work to load the graph and embeddings in CorGIE)?