Mixed-Initiative Methods
for Following Design Guidelines in Creative Tasks

Aditya Bharadwaj

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

T. M. Murali, Co-chair
Kurt Luther, Co-chair
Chris North
B. Aditya Prakash
Scott Klemmer

July 29, 2020
Blacksburg, Virginia

Keywords: Graph drawing, crowdsourcing, creative work, mixed-initiative systems,
checklists, feedback

Copyright 2020, Aditya Bharadwaj
Mixed-Initiative Methods
for Following Design Guidelines in Creative Tasks

Aditya Bharadwaj

(ABSTRACT)

Practitioners in creative domains such as web design, data visualization, and software development face many challenges while trying to create novel solutions that satisfy the guidelines around practical constraints and quality considerations. My dissertation work addresses two of these challenges. First, guidelines may conflict with each other, creating a need for slow and time-consuming expert intervention. Second, guidelines may be hard to check programmatically, requiring experts to manually use multipage style guides that suffer from drawbacks related to searchability, navigation, conflict, and obsolescence. In my dissertation, I focus on exploring mixed-initiative methods as a solution to these challenges in two complex tasks: biological network visualization where guidelines may conflict, and web design where task requirements are hard to check programmatically.

For biological network visualization, I explore the use of crowdsourcing to scale up time-consuming manual layout tasks. To support the network-based collaboration required for crowdsourcing, I first implemented a system called GraphSpace. It fosters online collaboration by allowing users to store, organize, explore, lay out, and share networks on a web platform. I then used GraphSpace as the infrastructure to support a novel mixed-initiative crowd-algorithm approach for creating high-quality, biological meaningful network visualizations. I also designed and implemented Flud, a system that gamifies the graph visualization task and uses flow theory concepts to make algorithmically generated suggestions more readily accessible to non-expert crowds. Then, I proposed DeepLayout, a novel learning-based
approach as an alternative to the non-machine learning-based method used in Flud. It has the ability to learn how to balance complex conflicting guidelines from a layout process. Finally, in the domain of web design, I present a real-world iterative deployment of a system called Critter. Critter augments traditional quality assurance techniques used in structured domains, such as checklists and expert feedback, using mixed-initiative interactions. I hope this dissertation can serve to accelerate research on leveraging the complementary strengths of humans and computers in the context of creative processes that are generally considered out of bounds for automated methods.
Mixed-Initiative Methods
for Following Design Guidelines in Creative Tasks

Aditya Bharadwaj

(GENERAL AUDIENCE ABSTRACT)

Practitioners in creative domains such as web design, data visualization, and software development face many challenges while trying to create novel solutions that satisfy the guidelines around practical constraints and quality considerations. My dissertation work addresses two of these challenges. First, sometimes the guidelines may conflict with each other under a certain scenario. In this situation, tasks require expert opinion to prioritize one guideline over the other. This dependence on expertise makes the design process slow and time-consuming. Second, sometimes it is difficult to determine which guidelines have been fulfilled. In this scenario, experts have to manually go through a list of guidelines and make sure applicable guidelines have been successfully applied to the final product. However, using a list of guidelines has its own drawbacks. Not all guidelines are applicable to a project, and finding a relevant guideline can be strenuous for experts. Moreover, a design process is not as simple as following a list of guidelines. Design processes are dynamic, non-linear, and iterative. Due to these reasons, a simple list of guidelines does not align with the designers’ workflow. My dissertation focuses on exploring mixed-initiative methods where computers and humans collaborate in a tight feedback loop to help follow guidelines. To this end, I present solutions for two complex creative tasks: biological network visualization where we can compute how well a design adheres to the guidelines but guidelines may conflict and web design where task requirements are hard to check programmatically. I hope this dissertation can serve to accelerate research on leveraging the complementary strengths of humans and computers in
the context of creative processes that are generally considered out of bounds for automated methods.
I dedicate my thesis to my parents: Dr. Chandresh Chandra Sharma and Mamta Sharma. Thank you for your immeasurable sacrifices and constant encouragement for my education. Without your love and blessings, I wouldn’t have been able to pursue a Ph.D.
I am deeply grateful to my family, friends, advisors, committee members, and many others not specifically named, without whom the work described in this dissertation would not have been possible.

First and foremost, I would like to thank my research advisors, Dr. Kurt Luther and Dr. T. M. Murali. Without their assistance and dedicated involvement in every step throughout my Ph.D., this dissertation would have never been accomplished. Their words of wisdom and guidance has brought me where I am today. Thank you for always being patient and supportive throughout the last five years.

Many thanks to my dissertation committee members, Dr. Chris North, Dr. B. Aditya Prakash, and Dr. Scott Klemmer, who gave me valuable feedback and supported me at different times during my dissertation research.

I would like to thank Divit P. Singh who is the joint first author of GraphSpace work discussed in Chapter 3 of my dissertation. GraphSpace was first introduced as part of his Master’s thesis. After his graduation, I was able to continue the work on GraphSpace and added multiple new features and a revamped easy-to-use web interface that finally led to a publication in Bioinformatics journal. Chapter 3 discusses the updated implementation and user workflow of GraphSpace since Divit’s Master’s thesis.

I also want to thank my collaborators, Pao Siangliulue, Adam Marcus, David Gwizdala, and Yoonjin Kim, whose useful insights and invigorating discussions helped shape some of the research presented in this dissertation.

I am also grateful to my labmates from both Murali Group and Crowd Lab. They provided me with useful inputs on my studies and best possible environment to study and conduct
research. Their valuable feedback on my practice talks helped me effectively present my research to the broader audience.

I would also like to express my sincere gratitude to my sister, Dr. Aditi Sharma, for providing me with unfailing support and constant motivation throughout my studies. This accomplishment would not have been possible without her. Thank you.

Last but not the least, my pillar of strength, my wife Prakriti, I am grateful for your steady support, constant encouragement, and bottomless patience throughout the last five years. Thank you for understanding my idiosyncrasies and keeping me cheerful through thick and thin.
## Contents

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

List of Tables .................................................... xxi

1 Introduction ....................................................... 1

1.1 Challenges and Opportunities for Research .......................... 3

1.1.1 Biological Graph Visualization .................................. 4

1.1.2 Web Design .................................................. 6

1.2 Research Questions .............................................. 7

1.2.1 Biological Graph Visualization .................................. 7

1.2.2 Web Design .................................................. 8

1.3 Thesis Statement ................................................ 8

1.4 Research Overview ............................................. 8

1.5 Main Contributions ............................................. 13

1.6 Outline of this Document ..................................... 14

2 Review of Literature ............................................. 16

2.1 Design Guidelines ............................................. 16

2.2 Graph Visualization ............................................. 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 Platforms for Network Analysis and Visualization</td>
<td>18</td>
</tr>
<tr>
<td>2.2.2 Algorithms for Network Layout</td>
<td>19</td>
</tr>
<tr>
<td>2.2.3 Machine Learning Approaches for Network Layout</td>
<td>21</td>
</tr>
<tr>
<td>2.2.4 Leveraging Human Abilities through Crowdsourcing for Network Layout</td>
<td>22</td>
</tr>
<tr>
<td>2.2.5 Combining Algorithms with Crowdsourcing for Network Layout</td>
<td>24</td>
</tr>
<tr>
<td>2.2.6 Leveraging Task Modes to Support Network Layout Tasks</td>
<td>26</td>
</tr>
<tr>
<td>2.2.7 Leveraging Scoring Systems to incentivize Crowds</td>
<td>27</td>
</tr>
<tr>
<td>2.3 Web Design</td>
<td>27</td>
</tr>
<tr>
<td>2.3.1 Checklists and Todo-lists to Support Web Design</td>
<td>28</td>
</tr>
<tr>
<td>2.3.2 Expert Feedback for Web Design</td>
<td>28</td>
</tr>
<tr>
<td>2.3.3 Automated Quality Assurance for Web Design</td>
<td>29</td>
</tr>
<tr>
<td>3 GraphSpace</td>
<td>30</td>
</tr>
<tr>
<td>3.1 Motivation</td>
<td>31</td>
</tr>
<tr>
<td>3.2 GraphSpace: System Description</td>
<td>32</td>
</tr>
<tr>
<td>3.2.1 Sharing and Organization</td>
<td>34</td>
</tr>
<tr>
<td>3.2.2 Searching across Networks and within a Network</td>
<td>34</td>
</tr>
<tr>
<td>3.2.3 Creating, Saving and Sharing Layouts</td>
<td>36</td>
</tr>
<tr>
<td>3.2.4 Filtering Network Nodes and Edges</td>
<td>42</td>
</tr>
<tr>
<td>3.2.5 Tags</td>
<td>44</td>
</tr>
</tbody>
</table>
3.2.6 Public Networks ........................................ 44
3.2.7 Uploading Networks ..................................... 44
3.2.8 Programmatic Access .................................... 47
3.3 GraphSpace Implementation ................................. 47
3.4 GraphSpace Use Cases ...................................... 48
3.5 Chapter Summary ........................................... 49

4 Flud: a hybrid crowd-algorithm approach .......................... 51

4.1 Motivation .................................................. 51
4.2 Description of the Flud System ................................. 52
  4.2.1 Requester Interface ...................................... 53
  4.2.2 Worker Interface ........................................ 56
  4.2.3 Layout Scoring System ................................... 63
  4.2.4 Recruiting and Training Crowd Workers .................. 65
  4.2.5 Simulated Annealing ..................................... 66
  4.2.6 Implementation Details .................................. 68
4.3 Evaluation .................................................. 68
  4.3.1 Networks ............................................... 69
  4.3.2 Crowd Workers .......................................... 69
  4.3.3 Crowd Worker Task ...................................... 69
4.3.4 Compensation ................................................................. 70
4.3.5 Experiment Design ............................................................ 70
4.4 Results .................................................................................. 78
  4.4.1 Q1: How should Flud assign criterion-specific modes to crowd workers 
           to optimize the scores they achieve? ........................................ 78
  4.4.2 Q2: How do the layout scores achieved by hybrid or crowdsourcing 
           approaches in Flud compare to automated methods? ................. 80
  4.4.3 Q3: Can users with biological training better understand Flud layouts 
           compared to those created by automated methods? .................... 85
  4.4.4 How did the crowd workers optimize the layout scores in the hybrid 
           and crowdsourcing approaches? ................................................. 91
  4.4.5 How did the criterion-specific clue impact use worker’s performance? 97
4.5 Discussion ............................................................................ 98
  4.5.1 Flud Facilitates Easy to Comprehend Network Visualizations .......... 98
  4.5.2 Flud Outperforms Algorithms in Network Visualization Tasks .......... 99
  4.5.3 Reflecting on Design Decisions ............................................. 101
  4.5.4 Design Implications .......................................................... 103
  4.5.5 Limitations ........................................................................ 105
4.6 Chapter Summary ................................................................... 106

5 DeepLayout: Laying Out Graphs Using Reinforcement Learning ....... 108
5.1 Motivation ................................................................. 108
5.2 The Graph Layout Optimization Problem ....................... 109
5.3 Layout Quality Function ............................................... 110
5.4 DeepLayout Approach .................................................. 112
  5.4.1 Formulating Graph Layout as Reinforcement Learning .... 113
  5.4.2 Rewards .............................................................. 115
  5.4.3 Observations ......................................................... 116
  5.4.4 Actions ............................................................... 118
  5.4.5 Architecture ......................................................... 118
  5.4.6 Training Method ..................................................... 119
5.5 Evaluation ................................................................. 121
  5.5.1 Measuring Structural Similarity between Graphs ............. 121
  5.5.2 Generation of Graph Datasets ..................................... 122
  5.5.3 Baseline Methods ................................................... 123
  5.5.4 Model and Training Configurations ............................. 124
  5.5.5 Implementation ..................................................... 124
  5.5.6 Results .............................................................. 125
  5.5.7 Training with Larger Datasets ................................... 128
  5.5.8 Impact of Hyper-parameters Choices .......................... 129
  5.5.9 Cost Function Design ............................................. 131
6 Critter: Augmenting creative work

6.1 Motivation .......................................................... 136
   6.1.1 A Motivating Example: Website Design .................. 138

6.2 Pilot Study: A Traditional Checklist .......................... 139
   6.2.1 Challenges and Drawbacks ............................... 139

6.3 Critter System Description ..................................... 141
   6.3.1 Dynamic Checklists ..................................... 141
   6.3.2 AutoQA .................................................. 143
   6.3.3 Contextual Reviewer-provided Feedback .................. 144

6.4 Evaluation ....................................................... 145
   6.4.1 Limitations .............................................. 147

6.5 Q1: How did designers’ use of Critter affect the quality of their designs? .... 148
   6.5.1 Dynamic Checklists ..................................... 149
   6.5.2 AutoQA .................................................. 151
   6.5.3 Reviewer Feedback ...................................... 152
   6.5.4 Usage Model ............................................. 152

6.6 Q2: How did designers feel about using Critter? ................... 153
6.6.1 Overall Experience ........................................ 154

6.6.2 Dynamic Checklists ....................................... 154

6.6.3 AutoQA ...................................................... 155

6.6.4 Reviewer Feedback and Recommendations ................. 155

6.7 Discussion and Conclusion .................................... 156

6.7.1 Implications for Practice .................................. 156

6.7.2 Alternative Design Opportunities .......................... 158

6.7.3 Generalization to other Domains .......................... 159

6.8 Chapter Summary ............................................. 160

7 Conclusions ..................................................... 162

7.1 Addressing the Research Questions ............................ 162

7.1.1 RQ1: How can we design a single system with the ability to store, organize, share, explore, and lay out graph visualizations? .... 162

7.1.2 RQ2: How can we use a mixed-initiative approach to enforce design guidelines in graph visualizations? .................. 164

7.1.3 RQ3: How can we design a computer agent with the ability to learn how to enforce design guidelines in graph visualizations? ..... 166

7.1.4 RQ4: How can we use mixed-initiative approach to enforce design guidelines in web design task? .......................... 168

7.2 Broader Implications ........................................... 169
List of Figures

3.1 A typical GraphSpace workflow. ............................................. 33
3.2 Screenshots of GraphSpace search functionality on the Graphs page. .... 35
3.3 Screenshot of edge search functionality on the Graph page. .............. 36
3.4 Screenshot of change layout panel on the Graph page. ..................... 37
3.5 Screenshot of node editor panel before making a change in node(s) style. .... 38
3.6 Screenshot of node editor panel after making a change in node(s) style. .... 39
3.7 Screenshot of edge editor panel before making a change in edge(s) style. .... 40
3.8 Screenshot of node editor panel after making a change in edge(s) style. .... 40
3.9 An example of layout editor workflow on GraphSpace. ..................... 41
3.10 Screenshot of filter panel on the Graph page. .............................. 42
3.11 Screenshot of filter panel when only nodes and edges with “k” ≤ 100 are visible. 43
3.12 Screenshot of filter panel when only nodes and edges with “k” ≤ 10 are visible. 43
3.13 Screenshot of “Graph Information” tab on the Graph page. ............... 45

4.1 Five layout criteria used in Flud to allow effective interpretation of signaling flow from source nodes (green triangles, placed at the top of the layout) to target nodes (yellow squares, placed at the bottom of the layout). .......... 54
4.2 Screenshot of Flud’s worker interface in the “Downward Pointing Paths” mode. 56
4.3 Illustration of downward and upward pointing edges. ............................... 59
4.4 Illustration of downward-pointing paths. .................................................. 59
4.5 Screenshots of the network visualization and the clue in the “Downward point-
ing paths” and “Non-crossing edge pairs” modes. ........................................ 61
4.6 Screenshots of the network visualization in Edge length, Node distribution,
and Node edge separation mode. ................................................................. 62
4.7 Figure illustrating the Crowd, Crowd-Random, and Hybrid (Crowd-SA) ap-
proaches evaluated in Chapter 4. ............................................................... 71
4.8 Figure illustrating the three simulated annealing schedules used in our hybrid
approaches. ................................................................................................. 72
4.9 Screenshots of the interface used in task-based effectiveness evaluation de-
scribed in Experiment 3. ........................................................................... 76
4.10 Comparison of scores achieved by crowd workers when modes are assigned
randomly (Crowd-Random) and in decreasing order of priorities (Crowd). .... 78
4.11 Distributions of the total scores of layouts created by different approaches. .. 80
4.12 Distributions of the number of downward pointing paths in the layouts created
by different approaches. ............................................................................... 81
4.13 Distributions of the percentage difference between median per-criterion scores
achieved by the Crowd approach in comparison to other methods. ............... 83
4.14 Bar plot of the average improvement in the total score per minute achieved
by a crowd worker or a simulated annealing run in the SA, Crowd, and Hybrid
approaches. .................................................................................................. 84
4.15 Performance results from task effectiveness evaluation described in Experiment 3 (Section 4.3.5). ................................................................. 85

4.16 Layouts of network G2 at six different stages in an example task sequence, illustrating how crowd workers improve the layout. ......................... 92

4.17 Bar plot showing average improvement in total score made by simulated annealing and crowd workers in hybrid approach. ......................... 94

4.18 G3 network layouts at four different stages in an example task sequence, illustrating how crowd workers and simulated annealing build on each other’s progress with the Crowd-SA100 approach. ............................ 96

4.19 Scores achieved by each approach over time for network G3 for a representative sequence. ................................................................. 97

5.1 Overview of reinforcement learning approach used in DeepLayout. .... 112

5.2 Policy and value network architecture used in DeepLayout. .......... 119

5.3 Layouts generated by DeepLayout, SA, and Dig-Cola, respectively for a representative graph in the add–1 dataset. ........................................ 128

5.4 DeepLayout’s performance on three different training datasets (del–1, add–1, flip–1) trained by varying different hyper-parameters. ............... 129

6.1 Critter components: Dynamic Checklists, AutoQA, & Checklist item-specific feedback ................................................................. 142

6.2 Bar plot of mean values for user engagement measures and mistakes defined in Section 6.5. ................................................................. 145
6.3 Bar plot of the average number of missed incomplete/complete design guidelines. 150

6.4 Bar plot visualization of the average rating given by the designers on a five-point scale where 1 represents poor and 5 represents excellent user experience. 153

A.1 The highest-scoring layouts generated by Flud and baseline methods for network G1. .............................. 222

A.2 The highest-scoring layouts generated by IPSEP-Cola and CerebralWeb baseline methods for network G1. .................... 223

A.3 The highest-scoring layouts generated by Flud and baseline methods for network G2. .............................. 224

A.4 The highest-scoring layouts generated by IPSEP-Cola and CerebralWeb baseline methods for network G2. .................... 225

A.5 The highest-scoring layouts generated by Flud and baseline methods for network G3. .............................. 226

A.6 The highest-scoring layouts generated by IPSEP-Cola and CerebralWeb baseline methods for network G3. .................... 227

A.7 Distributions of EC, EL, ND, and NED per-criterion scores of layouts created by different approaches. .................... 228

A.8 Plots of the time taken by three game sequences of different approaches to increase the total layout score. .................... 229

A.9 Plots illustrating the impact of criterion-specific modes and clues. .................... 230
List of Tables

1.1 Comparison of the creative tasks used for studies in this thesis. 7

4.1 Networks used for evaluation. 69

5.1 The layout methods and parameter ranges used for producing layout for graphs in testing datasets. 123

5.2 Comparison of the average best scores achieved by DeepLayout, SA, Dig-Cola, and IPSEP-Cola for 12 synthetically generated graph datasets. 126

5.3 Comparison of DeepLayout versus Simulated Annealing (SA). 127
Chapter 1

Introduction

Creativity is defined as the process of producing work that is both novel and appropriate [111]. While creating an artifact that is original and unexpected is essential in a creative task, novelty alone is not enough. Ensuring that the artifact conforms to task requirements and is judged useful and appropriate are important as well. However, if the requirements are too narrow and constraining, the set of possible solutions will be too limited for the work to be considered novel and original. Therefore, the choice of task requirements is crucial for generating creative artifacts that are both novel and appropriate.

To ensure the quality and appropriateness of creative artifacts, fields such as user interface design and software engineering gravitate toward best practices such as design guidelines [92, 113] or coding style guides [210]. Such guidelines act as quality constraints in creative tasks, and the people performing them have to balance their creative expression with practical requirements and quality considerations. For example, in order to consider drawings of some types of biological networks useful and appropriate, designers have to keep in mind the cellular locations of the nodes [60, 185] along with common guidelines for improving readability [197] such as fewer edge crossings and node overlaps. Similarly, web designers have to balance delivering websites that are aesthetically compliant with customers’ requests along with common best practices such as accessibility and responsivity. Becker et al. [16] discusses how task guidelines and conventions help practitioners achieve this balance in design work. Guidelines also facilitate efficient coordination of activity among practitioners.
for quick decision-making. However, following guidelines in complex creative tasks poses multiple challenges. Guidelines do not exist in isolation and may conflict with each other. A small change to the design to accommodate a particular guideline may decrease conformity to other guidelines. Automated methods have been proposed to help creators follow guidelines, but they are often limited to heuristic solutions \[194\]. Additionally, these solutions require a careful review by a domain expert. The expert must prudently review and choose to prioritize the guidelines which are most important for a designed artifact’s usability \[188\]. However, depending on experts to apply guidelines can be a slow, expensive process that does not scale well. Furthermore, in some domains, it can be challenging to simply encode guidelines in software, making it hard to automatically enforce or check for evaluation.

A majority of systems that support design guidelines in a creative process uses the ‘Human-Initiative’ approach \[41\] to address some of these challenges. In these type of systems, humans are the deciding agent and primary creator, whereas computers are merely support tools \[180\]. An example is a graph drawing interface in CrowdLayout \[186\] that specifies several design requirements along with layout tools for designers to use while drawing graphs. While it offers creative freedom, the human-initiative approach lacks the consistency and scalability offered by the ‘Computer-Initiative’ approach \[41\], where computers are the creators. An example of a computer-initiative approach would be a graph drawing algorithm such as force-directed layout \[61\] or circular layout \[66\]. Automated methods are fast and scalable at ensuring conformity to design constraints. However, they lack the ingenuity to effectively resolve conflicting constraints and often depend on heuristic solutions \[51, 207\]. More importantly, the inability to capture conflicting constraints can make the automatically generated artifacts inappropriate for practical use.

In recent years, we have seen the emergence of an intermediate ‘Mixed-Initiative’ approach
1.1 Challenges and Opportunities for Research

[41, 96, 221, 222] as a solution, where computer and humans collaborate in a tight feedback loop. Allen et al. [4] describes such a collaboration as a dialogue between humans and computers assigned to perform a task as a team. The communication between them can manifest in many ways. A common strategy is ‘fixed-subtask mixed-initiative,’ where the humans and computers take turns to solve separate pre-defined subproblems [187]. In other types of mixed-initiative interactions, the computers can also actively look out for critical scenarios and initiate a dialogue with the humans to clarify and correct their actions [75, 96]. Finally, in the ‘negotiated mixed-initiative’ type of interaction, there is no predefined assignment of sub-problems, and humans and computers negotiate to take the initiative. Overall, these rich set of hybrid interactions offer to solve problems more efficiently and effectively than either human- or computer-initiative [97] interactions individually. In this dissertation, I am particularly interested in the task of creating artifacts via such mixed-initiative interactions.

1.1 Challenges and Opportunities for Research

While mixed-initiative interactions have proved to be a useful approach for many problems, applying them to following design guidelines in a creative task is not straightforward. In my dissertation, I study mixed-initiative methods for following task guidelines in the context of two different types of creative domains: biological graph visualization and website design. Both domains have complex task guidelines that pose unique challenges (Table 1.1).
1.1.1 Biological Graph Visualization

Many fields of science require meaningful and visually appealing layouts of network (graph) data. A prominent example is the discipline of network biology, where scientists use networks to understand the chemical reactions and protein interactions that underlie processes that take place in the cell [11]. In order to present and analyze these networks, researchers require aesthetic layouts of these networks that clearly convey the relevant biological information. A graph visualization is considered aesthetic if it conforms to well-accepted principles of readability [197], such as minimizing edge crossings and the total length of edges. A layout of a network places nodes and edges strategically in a visual space to create a representation that satisfies certain practical and aesthetic constraints. Nearly every publication that uses network analysis includes a graph layout where the nodes and edges are arranged in two dimensions.

In the creative task of visualizing biological network data, the primary challenge is that graph drawing guidelines are known to often conflict with each other [18]. Prior work shows that these guidelines can be encoded quantitatively as metrics [155]. As a result, there exist many algorithms that aim to create layouts that optimize an energy-based objective function that encodes some of these guideline-based metrics [73, 147, 195]. However, in the presence of conflicting guidelines, the automated methods fail to capture complex constraints leading to poor-quality visualizations. Consequently, it is common for biologists to spend two to three hours manually improving each automatically generated visualization. Cytoscape [190] is a popular tool used by researchers to first lay out graphs using automated methods and then manually improve them. Unfortunately, this type of human-algorithm collaboration does not scale well for computational biologists who may generate hundreds or thousands of networks as part of their research.
1.1. Challenges and Opportunities for Research

In recent years, crowdsourcing has emerged as a scalable solution for businesses and researchers to rely on novices to produce expert-quality work [19, 23, 36, 112, 117, 118, 134]. Therefore, computer science researchers are now interested in designing systems and techniques that leverage crowd work in different domains [43, 224] such as graph visualization [185, 209, 226]. Fortunately, the number of guidelines (usually < 10) and design elements (nodes, edges) in the graph layout tasks are small [18, 38, 61, 154, 156, 209]. This affords us the opportunity to train non-expert crowd workers to perform manual tasks in graph drawing in a reasonable amount of time. Several systems provide an interface for network analysis and visualization such as BiNA [72], Cytoscape [58, 177, 189], HyperScape [35], Gephi [13], GraphViz [54], NetBioV [204] Pajek [39], and VisANT [99]. Despite these advances, these systems (at the time) did not provide an interface or infrastructure to allow network-based collaboration (store, organize, share, explore, and lay out graph drawings) on a single platform. Furthermore, it is unclear how to design interfaces and tasks that could allow novice crowds—who lack expertise in biology and computer science in general—to make domain-specific improvements to network layouts and balance different aesthetic constraints. The research in my dissertation seeks to bridge this fundamental gap.

According to Ferguson et al. [55], for a mixed-initiative system to be truly collaborative, the computer agent in a mixed-initiative system should have the capability to learn new ways to perform a task. With the recent advancements in Artificial Intelligence (AI), I see an opportunity to learn from a human’s design process in order to build a computer agent that can (semi-)autonomously produce visualizations that adhere to the guidelines. That said, the use of AI for graph drawing is a relatively new research paradigm and it is unclear how we can design a computer agent with the ability to learn from a layout process and enforce conflicting guidelines in graph visualization. My dissertation seeks to explore these challenges and opportunities to help scientists create high-quality network visualizations for
their research.

1.1.2 Web Design

In a graph layout task, guidelines can be captured quantitatively as a metric, and the number of guidelines is typically small (often less than 10), so searchability and navigation of guidelines is not a major issue. However, many creative domains such as web design, writing, poster design, or software programming have numerous (typically greater than 50) guidelines. The guidelines in these domains often require human judgement to check if they have been adequately satisfied [16] as they are hard to check programmatically. For example, one of the guidelines requires buttons to be linked to the page described in the customer requirements. This guideline is hard to check automatically because of the diverse and unpredictable nature of client needs. Checklists are effective in facilitating the fulfillment of such guidelines in domains with semi-structured tasks like aviation, automobile engineering, and construction [70]. However, it remains unclear how we can use these checklists for creative tasks. In practice, to tackle the large number of guidelines, web designers often use multi-page style guides while designing websites [92, 113]. Similarly, in software engineering, programmers often use coding style guides [210] while writing code. However, multi-page guidelines have multiple inherent usability drawbacks related to searchability, conflict, and obsolescence [203]. Despite these challenges, designers have to deliver novel work which satisfies the practical constraints and quality considerations. To my knowledge, there has been little exploration of methods to effectively incorporate the guidelines into the design process without disrupting the creativity of designers [113].
1.2. Research Questions

I propose the following four research questions to address challenges to integrating a mixed-initiative approach for applying design guidelines in two different types of complex creative tasks.

### 1.2.1 Biological Graph Visualization

**RQ1:** How can we design a single system with the ability to store, organize, share, explore, and lay out graph visualizations?

**RQ2:** How can we use a mixed-initiative approach to enforce design guidelines in a graph visualization task?

**RQ3:** How can we design a computer agent with the ability to learn to enforce design guidelines in a graph visualization task?
1.2.2 Web Design

**RQ4:** How can we use a mixed-initiative approach to enforce design guidelines in a web design task?

1.3 Thesis Statement

Task guidelines in creative domains often conflict with each other and are sometimes hard to check programmatically due to dynamic, project-specific requirements. Humans are good at resolving conflicts and adapting to contextual needs. However, they lack the speed and scalability of automation to search the solution space extensively. Furthermore, novices are prone to unproductive behavior like focusing on guidelines based on personal preferences instead of using the recommended priorities. On the other hand, computers can offer instantaneous feedback in well-defined scenarios, heuristic strategies for extensively searching the local area of a solution space, and automated prioritization. My thesis statement is that a mixed-initiative approach allows us to generate better creative artifacts than humans or computational methods alone because it can leverage the complementary strengths of human intelligence and automated methods to follow task guidelines. In this thesis, I present mixed-initiative methods for following guidelines in two different types of complex creative domains: biological graph visualization where guidelines often conflict (RQ1–RQ3) and website design where most of the guidelines are hard to check programmatically (RQ4).

1.4 Research Overview

In this section, I provide an overview of how I address these research questions in my thesis.
In order to answer the first research question, I propose and implement a system called GraphSpace. GraphSpace is a web-based platform for storing, organizing, sharing, exploring, and laying out networks. GraphSpace is currently used by around 500 users in their network biology projects. It assists in accelerating all aspects of network-based collaboration among computational biologists and experimentalists, including preliminary investigations, manuscript development, and dissemination of research. This work later serves as the foundation to support the tools and experiments proposed to address research questions 2 and 3.

In my second work, I propose a novel hybrid crowdsourcing approach for creating biological graph layouts to address the second research question. In this approach, I leverage the combination of 1) spatial reasoning skills and intelligence of humans to observe flow, crossings, and distances, with 2) the fine-tuning capabilities of a Simulated Annealing algorithm to draw graph layouts that meet the specified criteria. Specifically, my approach uses a sequential and collaborative process wherein crowd workers and a Simulated Annealing-based layout algorithm [38] build on each other’s progress. The Simulated Annealing algorithm seeks to iteratively reach a layout with a high score by randomly searching the neighborhood of an initial guess. Its strength lies in being to avoid getting stuck at local optima — solutions that are better than the others nearby but are not the very best — by probabilistically accepting inferior solutions. In contrast, crowd workers use their intelligence and creativity to optimize the score by making modifications that may be global in nature and challenging to achieve using an automated method.

To facilitate the crowd task in this collaborative process, I develop Flud, a crowd-powered system that allows novice humans to design biologically meaningful network layouts with the help of algorithmically generated suggestions. Flud uses GraphSpace as the backend infrastructure for storing, exploring, and interpreting graph layouts. In Flud, the goal of
crowd workers is to create a layout that optimizes a weighted layout quality function based on the pre-specified design criteria. I ask the crowd workers to generate layouts that satisfy five layout criteria that are known to conflict with each other. These criteria include four previously defined aesthetic considerations \([38, 154, 155]\): 1) minimizing the number of edge crossings, 2) keeping nodes connected by an edge close to each other, 3) dispersing disconnected node pairs, and 4) increasing the separation between nodes and edges. I also introduce a new, fifth criterion inspired by a biological application to cellular signaling: 5) maximize the number of downward pointing paths in the layout.

To evaluate Flud and the novel mixed-initiative layout approach, I recruit nearly 2,000 novice crowd workers on Amazon Mechanical Turk to lay out and visualize complex protein networks that represent signaling pathways in human cells. I create three different versions of the hybrid crowd–algorithm approach and evaluated them against a crowd-only strategy and algorithmic baselines. I conduct this evaluation in two parts. In the first part, I observe that novice workers are inclined to select task modes based on personal preferences instead of task priorities. The problem with this behavior is that, in the presence of conflicting guidelines, the order of the task modes plays an important role. I empirically demonstrate that randomly assigning task modes can lead to unproductive work. I also show that computationally assigning the task modes in the order of their priorities generates better results than random assignment. In the second part, I show that a collaboration between humans and algorithms can be effective in the network layout task. Specifically, the results indicate that such a collaboration leads to higher-scoring layouts than either from humans or from algorithms alone. I find that system features such as criterion-specific modes and clues supported crowd workers in the visualization tasks. I also show that the crowd workers who moved one of the nodes suggested by Flud contributed more to the overall layout score in comparison to rest of the nodes in the network.
Next, I conduct a user study to evaluate the task effectiveness \cite{164} of network visualizations generated using Flud versus algorithmic baselines. I recruit seven participants with a research background in computational biology to identify the information flow in the layouts from receptor nodes (source nodes) to transcription factor nodes (target nodes). The results show that participants were more accurate, faster, and more confident while performing these tasks with Flud-generated layouts in comparison to algorithmic baselines. I also observe that participants rated Flud-generated layouts as easier to use. Beyond task effectiveness, I also gain insight into the importance of the criterion related to downward pointing paths and why Flud layouts are easier to understand.

After demonstrating that mixed-initiative crowd–algorithm approach can effectively solve layout tasks, my next work addresses the third research question. It proposes an AI-based approach to replace the Simulated Annealing-based computer agent used in Flud’s hybrid approach. The limitation with Simulated Annealing is that it does not have the capability to learn to distinguish between good or bad layout updates for a given graph. Instead, it depends on randomized updates to correct earlier bad choices. As a solution, I propose DeepLayout, a reinforcement learning technique to find a graph drawing that optimizes a given layout quality function. DeepLayout addresses the above-mentioned limitation by learning an implicit representation of the layouts and a mapping from layouts to actions. These actions determine how DeepLayout positions the nodes iteratively. To the best of my knowledge, DeepLayout is the only graph layout approach with the ability to generalize, i.e., it has the ability to learn from a layout process to generate drawings for new, unseen graphs. Specifically, I demonstrate DeepLayout’s ability to learn how to optimize the same layout quality function that I utilize in Flud. Algorithmically optimizing this layout quality function is a challenging problem. Specifically, minimizing the number of edge crossings alone is NP-complete \cite{69}, and topologically laying out a network with cycles is also NP-hard \cite{108}.
To showcase DeepLayout’s ability to optimize the given layout quality function, I present an empirical study that evaluates its effectiveness and efficiency in drawing new graphs that are structurally similar to the training data. This study conducts an evaluation for 12 synthetic graph datasets. The graphs in each dataset are structurally similar to each other and generated from directed graphs representing signaling pathways in human cells. The results show that, on average, DeepLayout outperforms state-of-the-art methods such as Dig-Cola \[46\] and IPSEP-Cola \[48\] in all datasets, and the Simulated Annealing method in nine (75%) of the datasets. Furthermore, I show that DeepLayout was, on average, 23.6% faster than Simulated Annealing. I also use qualitative comparisons to demonstrate that the layouts created by DeepLayout are superior to those output by a Simulated Annealing-based approach and other state-of-the-art algorithms designed for drawing directed graphs.

Finally, to address the fourth research question, I investigate the iterative and real-world deployment of traditional quality assurance techniques like checklists, automated checkers, and reviewer feedback for a web design task. Checklists are an effective tool in adding structure and repeatability to complex processes, allowing organizations to enforce hard-to-encode guidelines on a per-project basis \[85\]. Unfortunately, performing creative tasks is not as simple as following a protocol. Creative processes are complex, dynamic, and non-linear in nature \[170, 201\]. Some aspects of design processes are also iterative, making the application of step-by-step checklists more challenging.

To explore their usage, I first conduct a needfinding study with expert web designers and identify five challenges faced by the designers. Informed by these findings, I create Critter, a mixed-initiative system with three components: Dynamic Checklists that progressively disclose guideline complexity with a self-pruning hierarchical view, AutoQA to automate common quality assurance checks, and guideline-specific feedback provided by a reviewer to highlight mistakes as they appear. Next, to evaluate Critter’s effectiveness, I conduct
an observational study with professional web designers creating websites for 30 real-world clients. In this study, I observe that the more engaged a designer was with Critter, the fewer mistakes they made in following design guidelines. Designers rated the AutoQA and contextual feedback experience highly, and provided feedback on the tradeoffs of the hierarchical Dynamic Checklists. I additionally observe that a majority of designers rated the AutoQA experience as excellent and felt that it increased the quality of their work. Finally, I discuss broader implications for supporting complex creative tasks with numerous, hard-to-encode task guidelines and non-linear completion paths.

1.5 Main Contributions

In summary, my dissertation makes the following contributions:

1. An open-source web-based platform, GraphSpace, that fosters team science by allowing collaborating research groups to easily store, interact with, lay out, and share networks.

2. A crowd-powered system, Flud, that combines crowdsourcing with computational methods to create high-quality visualizations of biological networks. The results from an experiment with around 2000 crowd workers and a mixed-methods study of participants with biological training show that:

   - Assigning the modes in the order of their priorities generates better results than assigning them randomly.
   - Collaboration between crowds and algorithms leads to higher scoring layouts than either from crowds or algorithms alone.
   - Layouts generated by the mixed-initiative crowds-algorithm approach supported significantly more accurate, faster, more confident, and easier interpretations of
information flow in biological networks compared to algorithmic baselines.

3. A graph layout approach, DeepLayout, that uses reinforcement learning to learn the layout strategy for efficiently optimizing a layout score/energy function based on the direction of the flow of information in the network, edge crossings, and relative distances between nodes and edges. An evaluation of 12 synthetic graph datasets generated from a biological network shows that DeepLayout outperforms the Simulated Annealing algorithm used in Flud’s mixed-initiative approach in 9 out of 12 datasets.

4. A system, Critter, that combines automation with human-driven planning and reflection to help experts navigate dynamic project-specific design guidelines. A mixed-methods evaluation of web designers as they create 30 websites for real-world customers showed that the more engaged a participant was with Critter, the fewer mistakes they made in following design guidelines.

1.6 Outline of this Document

Here, I describe the organization of the upcoming chapters in this dissertation. The research presented in Chapters 3 [183] and 6 [22] has been published, and the work in Chapters 4 and 5 is currently under review. Chapter 2 presents a literature review of methods for effectively applying design guidelines. It discusses the benefits and limitations of existing methods and how the solutions proposed in this dissertation differ from the prior work.

Chapter 3 addresses Research Question 1 by presenting a web-based platform called GraphSpace for storing, interacting, laying out, and share networks. The chapter discusses the features offered by GraphSpace and how users can interact with it. It also discusses the implementation details and use cases for GraphSpace. The GraphSpace platform was later used as the
1.6. Outline of this Document

infrastructure to support systems and experiments described in the subsequent chapters 4 and 5.

Chapter 4 presents a novel hybrid crowd-algorithm approach for managing conflicting and complex guidelines in biological network visualization, and a crowd-system called Flud facilitating this approach to address Research Question 2. The chapter discusses in detail the set of design criteria (guidelines) used for the layout task and implementation of Flud system. The chapter also describes the setup and results for the empirical studies conducted to showcase the effectiveness of Flud as a graph visualization system.

Chapter 5 addresses Research Question 3 with DeepLayout, a novel graph layout approach with the ability to generalize, i.e., it has the ability to learn from a layout process to generate drawings for new, unseen graphs. This chapter discusses the formulation of the reinforcement learning method for a graph layout problem, describes the deep neural network architecture, and reflects on different design choices in DeepLayout. The chapter also reports an evaluation study that compares the proposed approach to the state-of-the-art graph layout algorithms.

Chapter 6 addresses Research Question 4 with Critter, a mixed-initiative system that helps designers to follow guidelines in web design task with three components: Dynamic checklists, AutoQA, and guideline-specific feedback. The chapters discusses the implementation of each component in detail. It also reports a mixed-methods evaluation of web designers as they create 30 real-world websites for a web design company using Critter. In the end, the chapter discusses the broader implications for supporting complex creative tasks.

Finally in chapter 7, I conclude with a summary of lessons learned from the research questions, the broader implications and the future direction for research presented in this dissertation.
Chapter 2

Review of Literature

In this chapter, I review the prior work on design guidelines and methods used for enforcing them in graph visualization and web design tasks.

2.1 Design Guidelines

Design guidelines are sets of rules designers should follow to ensure that their artifacts are up to standards. They are widely developed and applied while designing user interfaces [74, 92, 113], websites [125], graph drawings [154, 155], interior decor [139], and software programs [210]. They can ensure the quality of end products and reduce stress of the designers [113]. Some of these guidelines are products of expert judgement, common sense, and practical experience [26] while others are derived from more rigorous testing and evaluation [125, 154].

However, design guidelines alone are not always effective. Unclear guidelines discourage designers from following them [202]. Conflicting design guidelines require expertise to resolve conflicts to benefit the overall design in the long term. A large number of guidelines makes it difficult for designers to find relevant guidelines [113, 142]. Further, new design guidelines that differ from designers’ previous experience are not effective even when the designers are motivated to follow them [203].
2.2 Graph Visualization

Prior work has explored various approaches to help designers follow guidelines. These approaches typically depend on the nature of guidelines. In tasks with computable guidelines such as graph visualization, approaches typically incline towards automation [67, 73]. One approach involves integrating the guidelines into the design tools by detecting or preventing deviations from guidelines. For example, Merrell et al. implemented a system that proposes alternatives that comply with the guidelines [139]. With the guidelines embedded in the system, the designers do not have to think about double-checking them, reducing the error in interpretation. This approach, however, only works well with guidelines that can be checked automatically. It is also less flexible because the guidelines are embedded tightly into the systems. A different approach just provides a list of guidelines [186], providing more flexibility with the choice of guidelines. However, such an approach is not effective while working with a large number of design guidelines, since searching for applicable guidelines can be strenuous. As a solution, some approaches focus on making it easy to find relevant guidelines from a collection by reorganizing them [125] or making them searchable [92]. This approach helps designers save time and mental effort in locating appropriate guidelines. Another approach adopts examples to clarify ambiguous guidelines [92]. In the later sections in this chapter, I discuss the different approaches I reviewed for effectively following design guidelines in graph visualization and web design tasks.

2.2 Graph Visualization

Graph layouts are widely used in many domains to visualize content underlying graph data. For example, biologists study information flow in cell signaling pathways and metabolic networks to understand the underlying mechanisms [11]. Similarly, ecologists look at pelagic food web networks to understand the mechanisms underlying the global degradation of
coastal ecosystems [212]. In this section, I review different graph visualization systems and techniques.

### 2.2.1 Platforms for Network Analysis and Visualization

Many systems have been developed to visualize and analyze networks. Prominent systems used by researchers include NetworkPainter [109], CellMaps [165], BiNA [72], Cytoscape [177, 189], Gephi [13], GraphViz [54], NetBioV [204] Pajek [39], and VisANT [99]. Each system provides a unique set of features that are useful for network-based collaboration. NetworkPainter [109] allows its users to manually create network diagrams and overlay flow cytometry data on them. However, it does not allow users to explore and organize their networks. CellMaps is a web tool that allows display, editing, exploring, and analysis of biological networks [165]. However, it does not appear to have features that permit sharing of networks and layouts on the CellMaps website. At the time of the research presented in this dissertation (specifically Chapter 3), graph visualization systems did not provide all features required for frictionless network-based collaboration. These requirements include storing, organizing, exploring, sharing and laying out networks.

The Cytoscape ecosystem provides solutions to some of the above-mentioned requirements. Cytoscape [189] itself is a popular and powerful software environment for visualizing and interacting with networks. CyNetShare (http://idekerlab.github.io/cy-net-share), a plugin in Cytoscape, provides a simple mechanism for a user of Cytoscape to export a network as a file and point the CyNetShare website to the location of this file, which allows users to share the URL of the network layout with collaborators. Since CyNetShare is designed for sharing networks one at a time, it does not readily scale to multiple networks. Cytoscape.js [58] is a library of graph algorithms that makes network analysis and
visualization accessible through web. However, so far Cytoscape.js has been used for only sharing networks on customized websites. NDex [151] is a repository designed for storing networks that supports many file formats, network queries, and the creation of groups for collaboration. However, unlike the GraphSpace system proposed in Chapter 3, NDex does not allow users to modify, save, or share network layouts. While my dissertation proposes the idea of groups and the notion of private collaboration vs. public sharing (Chapter 3) independently of NDex, the two resources are similar in this respect.

2.2.2 Algorithms for Network Layout

Many algorithms have been developed to aid the useful visual interpretation of graphs [42, 73, 147, 195]. In my dissertation, I focus on 2D layout methods that produce node-link diagrams. Popular 2D layout algorithms include circular [15, 66], orthogonal [53], and spectra [120, 121] algorithms. These techniques aim to achieve multiple aesthetic properties such as grouping connected nodes closer to each other, avoiding edge crossings, and decreasing node overlaps [154, 155]. Some methods such as force-directed algorithms [17, 61, 104] and spring electrical models [98] adopt the energy-based layout approach [73]. They aim to create layouts that optimize an energy-based objective function that encodes some of these aesthetic properties. Several of these methods can generate layouts in a few seconds for moderately-sized networks. Implementations for these algorithms is available in various tools or libraries, e.g., Cytoscape [178], Gephi [102], Graphviz [54], NetworkX [83], NetBioV [205], and Pajek [14]. However, these algorithms are designed to optimize their specific objective functions. They are not flexible enough to accommodate domain-specific constraints arising in a wide variety of applications. Most importantly, these methods are designed specifically with undirected graphs in mind.
Drawing directed graphs is a complex challenge since layouts need to showcase the connectivity between nodes and the directionality of the information flow through the connections (edges) themselves. Prior literature suggests that methods should consider aesthetic criteria like downward pointing edges for directed graphs [154]. Therefore, algorithms designed for drawing undirected graphs are inappropriate for graphs with directed edges [27, 47, 67]. In my dissertation, I present two novel approaches (Chapters 4 and 5) for laying out directed graphs, e.g., a biological signaling pathway.

Layout methods for directed graphs such as Sugiyama and the hierarchical algorithm in the Graphviz package *dot* usually lay out nodes in a level-wise manner [52, 67, 194]. Another such popular method is Dig-Cola [46], which, unlike Sugiyama, considers the nodes within a cycle to be at the same level of the hierarchy. CerebralWeb [60], a biological graph visualization tool, specifically uses the cellular locations of nodes to arrange them in layers such that their order resembles the typical receptor to transcription factor signaling flow in biological graphs. Some layout methods aim to satisfy constraints that require edges to be downward pointing [49, 172]. For example, IPSEP-CoLa [48] uses a force-directed approach to enforce the constraint that directed edges should point downward. Since highlighting information flow is an important criterion for experiments in this dissertation, I use Dig-Cola and IPSEP-Cola as baselines to compare performance against the approaches proposed in my dissertation.

Another popular graph drawing technique is the Simulated Annealing-based layout algorithm (SA) [38]. Simulated Annealing is a well-known approach that can be used to optimize any arbitrary, non-differentiable objective function. In a graph layout context, the algorithm iteratively updates a layout by making a small random change. It accepts the new layout if it achieves a better value for the objective function than the current layout. If the new layout is worse, the algorithm accepts it with a probability that depends on the difference
of the two values of the objective function. This strategy allows Simulated Annealing to perform an extensive search for global optima in the solution space. Due to its flexibility to accommodate any set of layout criteria, I used Simulated Annealing as an additional baseline method. More importantly, I also use Simulated Annealing as a computer agent in the hybrid crowd-algorithm approach proposed in Chapter 4. My dissertation shows that the hybrid generated better results than either from crowds or the Simulated Annealing algorithm alone. That said, the Simulated Annealing-based approach has a major limitation: it does not learn from its previous layout updates and depends on random changes to improve the layout quality. My dissertation addresses this limitation with a reinforcement learning-based approach that learns from a layout process and can efficiently lay out new graphs similar to the ones used for training (Chapter 5).

2.2.3 Machine Learning Approaches for Network Layout

The application of machine learning for network visualization can be roughly classified into four categories based on the visualization tasks being tackled. The first category trains models to evaluate the quality of the layouts [84] and discriminate between good and bad visualizations [119]. The second category of methods learns a user’s preference for aesthetic criteria in order to generate 2D layouts [9, 12, 137, 163, 191]. The third category includes methods that support the exploration of different aspects of layouts, including exploration of layout methods by finding topologically similar graphs and previewing their layouts generated by different methods [128], graph structure exploration by finding structurally similar subgraphs in a large network and drawing them [29], and training generative models to explore the diverse layout search space [127].

Finally, the fourth category presents methods that draw graph layouts for new, unseen
graphs. An example here is the use of t-distributed Stochastic Neighbor Embedding (t-SNE), a popular dimensionality reduction technique, to embed high dimensional data like graph-theoretic distances on to a 2D layout space [123]. Another example is DeepDrawing, which uses a graph-LSTM-based model to learn layout characteristics from a given set of layout examples and create drawings with similar properties for new graphs [216]. However, DeepDrawing and t-SNE do not consider the edge direction and are designed only for undirected graphs.

In my dissertation, I propose DeepLayout (Chapter 5), a reinforcement learning-based approach to generate drawings for new, unseen graphs. It also falls in the fourth category. However, my dissertation improves over techniques such as DeepDrawing and t-SNE in two important ways. First, it benefits by using reinforcement learning to balance between exploitation and exploration. Purely exploitative methods, e.g., supervised learning, try to replicate any behavior or property seen in training data. In contrast, exploration, allows the model to correct for bad behavior that it may have learned early during the training process. Second, my dissertation focuses on a layout constraint (maximizing downward pointing paths) that is dependent on the edge directions. Moreover, unlike DeepDrawing, DeepLayout does not have the overhead of finding layout examples for training purposes. Instead, DeepLayout only needs a cost function that encodes the desirable layout properties.

### 2.2.4 Leveraging Human Abilities through Crowdsourcing for Network Layout

The idea of using human computation to solve real-world scientific problems is not new [32, 56, 173]. It has been used to address a wide range of complex problems in biology, such as protein folding [33], RNA sequence design [131], DNA sequence alignment [112], molecular
2.2. Graph Visualization

design [7], and neuron reconstruction [114]. Inspired by the success of these systems, I designed Flud (Chapter 4) to leverage the creativity and cognitive abilities of humans to create biological network visualizations, a challenge that has not been previously tackled using human-computation systems.

Some research has examined how users lay out networks, including in comparison to automated layouts. Two studies [50, 209] asked participants to draw an aesthetic layout of a given network. Dwyer et al. [50] concluded that the best user-generated layouts performed as well as or better than automated layouts based on physical models. Although these studies do not use crowd workers, they provide evidence that non-expert humans can create network layouts comparable to algorithms.

Researchers have explored using non-expert volunteers (e.g., citizen scientists) to perform other visualization tasks. For example, systems such as ManyEyes [213] and Sense.us [91] have used novice crowds to create, collaborate on, and analyze data visualizations. Connect the Dots [132] and CRICTO [31] ask novice crowds to build social networks for intelligence analysis, but not to create layouts for them. Inspired by these projects, I explore the use of non-expert workers on Amazon Mechanical Turk to create biological network layouts in Chapter 4.

Most related to my dissertation, CrowdLayout leverages crowd workers to design layouts of biological networks that satisfy domain-inspired guidelines specified in natural language [185]. CrowdLayout also uses crowd workers to evaluate how well these layouts satisfy the guidelines. Work in CrowdLayout shows that translating domain knowledge into representational guidelines can allow non-expert crowds to create effective layouts.

My dissertation work (Chapter 4) differs from and complements these efforts in several important ways. First, I focus on combining multiple aesthetic criteria with a specific domain...
knowledge guideline, all of which can be translated into mathematical formulae. Second, I score layouts to provide real-time feedback to the user who is modifying the layout rather than (as in the case of CrowdLayout) wait for other crowd workers to evaluate the layout. Third, I formulate layout creation as a collaboration among multiple crowd workers or between crowd workers and computational engines, in contrast to single-user design tasks.

2.2.5 Combining Algorithms with Crowdsourcing for Network Layout

Prior work has explored using mixed-initiative systems to perform a variety of tasks like data wrangling [106], exploratory analysis [220], and natural language translation [77]. In a recent work, Heer [90] discussed how these systems use *shared representations* like text-editing interfaces and domain-specific languages through which humans and algorithms can work together to accomplish a goal. In contrast to these systems, the shared representation in Flud (Chapter 4) involves three components: (i) a network layout interface, (ii) 2D coordinates on the screen as potential actions, and (iii) a scoring scheme as a shared objective. More importantly, I use these shared representations to support mixed-initiative interaction in a novel network layout task.

Some mixed-initiative systems also leverage the complementary strengths of human intelligence and automated techniques to help users understand network data. These include a visualization tool that allows users to analyze legal citation networks with degree of interest functions [208], and a system that helps users explore clusters of similar research papers via machine learning and visualization [28]. These systems can be highly effective for individual expert users, but my dissertation (Chapter 4) considers how non-expert crowds can work with algorithms to pre-process network layouts for expert users.
Researchers have also begun exploring mixed-initiative systems that use crowdsourcing. For example, Flock [30] helps build accurate classifiers by supporting a mixed-initiative interaction where crowd nominates features and machine learning weighs them. Jellybean [169] was able to accurately count objects in images by combining results from crowds and computer vision algorithms. Ideahound [181] combines human judgment with machine learning techniques to create a computational semantic model of the emerging solution space. Mobi [229] demonstrated how a mixed-initiative crowdsourcing system could support trip planning by allowing a traveler to specify both quantitative global constraints (e.g., allotted time, number of activities) enforced by the software, and qualitative constraints (e.g., visiting the beach vs. the downtown) satisfied by crowd workers. Subsequent research explored these ideas for other tasks and domains, such as conference planning [116]. Moreover, while Mobi guided worker effort through todo items, systems like Flock, Jellybean, and Ideahound used simple instructions to guide them. In contrast, I consider how mechanisms such as task modes and clues can motivate desired behavior in crowd workers. Also, Flud differs in that my dissertation adapts these ideas for a novel task type (i.e., 2D network layout that adheres to multiple guidelines) and task domain (i.e., biology).

Yuan et al. [226] proposed a strategy that utilized crowd workers to lay out subnetworks of a complex network. Subsequently, the authors used a constrained distance-embedding algorithm to compose the layouts of the subnetworks into one for the entire network. In contrast, my dissertation seeks to facilitate the design process for crowd workers in order to improve the overall quality of layouts they generate.
2.2.6 Leveraging Task Modes to Support Network Layout Tasks

Task modes are commonly used in games to offer players different gameplay settings, difficulty levels, tools, rules, and even graphics. A typical example of a mode in a game is the choice between single-player versus multi-player setting. Modes are common in human-computation systems as well. For example, Phylo [112] has three different task modes (Story, Ribo, Phylo) with different task settings including one for RNA molecules. Foldit [33] has five task modes: Modeless, Pull, Structure, Note, Design. Each of these modes endows the citizen scientists with unique abilities, which are otherwise not available in other modes. In Flud (Chapter 4), my dissertation adapt these ideas and use task modes as a system feature to focus the crowd workers’ attention on one sub-task at a time. Specifically, my dissertation uses five modes where I enable crowd workers with a network visualization and clues unique to a given mode. However, the use of modes in my dissertation differs from other systems in that Flud intelligently allocates a mode to crowd workers instead of giving them an option to choose it. It allows the system to decide the primary focus of the crowd workers for a given task. This strategy contrasts the mixed-initiative approach in human-computation games like Foldit where humans have the ability to choose the mode and the task-specific algorithms. In other words, Foldit players control the computer agent’s focus for a given task. However, in my dissertation (Chapter 4), I observe that novices find it challenging to choose the appropriate criterion to focus on at a given stage in the task which leads to unproductive work. As a solution, in my dissertation (Chapter 4), I empirically study and present the best strategy to computationally assign modes to the crowd workers.
2.3. Web Design

2.2.7 Leveraging Scoring Systems to incentivize Crowds

Prior research in crowd-powered conversational assistants [100, 129] shows that points-based reward systems can motivate crowd workers to stay longer and reinforce productive behavior. Unlike these earlier efforts that used fixed reward amounts for each point earned by the crowd workers, Flud (Chapter 4) computes the reward amount dynamically, where the compensation increases as the highest score improves. This strategy is critical for Flud (Chapter 4) since the layout design task gets increasingly complex and challenging as the crowd workers improve the layout. Therefore, to keep the crowd workers interested in challenging tasks, my dissertation proposes to raise the compensation as the complexity increases.

The scoring system generally depends on more than one component. While systems such as Phylo [112] provide detailed scoring information by showcasing values for all of the components that make up the overall score, other approaches, e.g., Evorus [100], Chorus [129], Foldit [33] and EyeWire [115], only show the overall score. Since crowd workers need to consider multiple criteria while laying out the network, Flud (Chapter 4) shows all of the component scores along with the overall scores. Our scoring information differs from these systems in two important ways. First, to help crowds understand the scoring method, I transparently show the weights of all the component scores. Second, I use visual cues such as up/down arrows and green/red colors to allow crowds to track progress after each move.

2.3 Web Design

In this section, I review the literature for traditional quality assurance techniques used for maintaining quality standards and task requirements.
2.3.1 Checklists and Todo-lists to Support Web Design

Checklists have been shown to provide multiple benefits, including reminding users of critical steps, creating consistency, enforcing regulation of policies, and offering a framework for evaluation [176]. The primary purpose of many checklists is essentially quality assurance through error reduction and guideline adherence [70]. Checklist usage varies considerably by domain. For example, in the aviation industry, checklists are standardized and compulsory [40], and completion of checklists from memory is considered a protocol violation [85]. In contrast, design guidelines are self-imposed and fungible (e.g., it’s common for designs to hold aesthetics and structure at tension), reinforcing prior work that emphasizes the non-linearity of complex work [159]. In the Critter system proposed in my dissertation (Chapter 6), the checklists are dynamic and adapt to project requirements and prior experience. Additionally, in my dissertation I ask the experts to use the checklists at specificity level they see fit in their workflow.

With time, checklists have been digitized, showing advantages over traditional media. For example, the Boeing 777 Electronic Checklist, developed in the early 1990s, decreased errors by an additional 46% as compared to paper-based checklists alone [24]. My dissertation (Chapter 6) builds on this research by adding human-computer interactions like self-pruning and toggling that afford dynamicity to the checklists.

2.3.2 Expert Feedback for Web Design

Expert feedback is important in design tasks. It helps recipients to increase their knowledge [89], learn about best practices [93], produce better results [43], and avoid mistakes in the future [82]. It allows the recipients to reflect on their performance and identify their strengths and weaknesses [218]. A related thread of work considers how to provide effec-
tive feedback, especially to help novices. In general, good feedback is specific, actionable, and contains an explanation [144]. Kulkarni et al. used examples as feedback to improve creative work [124]. Yuan et al. used rubrics to structure design feedback [225]. However, in semi-structured tasks, human experts are susceptible to judgment errors [182], suggesting opportunities for mixed-initiative support tools. Specifically, experts like designers have a memory capacity and can forget a guideline while reviewing the work. My dissertation (Chapter 6) addresses this issue by using design guidelines as rubrics, ensuring consistent feedback supported by direct and actionable design guidelines.

2.3.3 Automated Quality Assurance for Web Design

An automated critique system or quality assurance checker scans the work submitted by the user for possible violations of design guidelines in a given field [57]. Automated quality assurance is a common practice in domains ranging from structured tasks like manufacturing [79] and medical device engineering [63] to creative tasks like writing [217], software development [45], and design [57]. Embedding such automated critics in work practice can help designers learn how to identify and solve problematic situations early in the design process [57]. In design, trust in such systems plays a critical role in its effectiveness. For example, a study found that designers rated a critiquing tool valuable, provided that they feel they are in control and the tool indicates the severity of the detected error [133]. The automated critic in my dissertation (Chapter 6), AutoQA affords designers the ability to catch errors based on design guidelines. In this dissertation, I study the relative benefits of automated critique, self-review, and reviewer feedback.
Chapter 3

GraphSpace: stimulating interdisciplinary collaborations in network biology

This work is based on the following published work in the journal *Bioinformatics* where I was joint first author with Divit P. Singh.


GraphSpace was first presented in Divit P. Singh’s Master’s thesis document. However, the GraphSpace system has since been significantly updated, including features like indexing capabilities, alias search, RESTful API, a Python client, support of Cytoscape.js style files, node and edge style editor, group invitation link, and a revamped easy-to-use user interface. This chapter discusses the updated GraphSpace interface and user workflow since Divit’s Master’s thesis.
3.1 Motivation

Networks have become ubiquitous in systems biology. Visualization is a crucial component in their analysis. Several systems implement methods for network analysis and visualization [35, 58], such as BiNA [72], Cytoscape [177, 189], Gephi [13], GraphViz [54], NetBioV [204], Pajek [39], and VisANT [99]. Despite these advances, collaborations within research teams in network biology are hampered by software systems that are either specific to a computational algorithm, create visualizations that are not biologically meaningful, or have limited features for sharing networks and visualizations. In this dissertation, I seek to explore and design mixed-initiative system for enforcing conflicting guidelines within the context of graph drawing. Most importantly, in order to truly harness the power of crowdsourcing we need a way to share, explore, interpret graph drawings over the internet. The current landscape of graph drawing tools do not satisfy these requirements. Therefore, in this chapter, I aim to address Research Question 1: How can we design a single system with the ability to store, organize, share, explore, and lay out graph visualizations?

To answer Research Question 1, I outline several challenges we aim to address:

**Layouts.** Most of the automated layout algorithms do not utilize any biological information underlying the networks. Thus, researchers need layout tools that can help them readily use their knowledge and intuition to modify node positions manually to bring out salient features.

**Sharing.** Computational biologists may generate hundreds or thousands of networks and layout that they seek to share without manual overhead. These networks should be viewable only to the collaborators.

**Organization.** A research group working on multiple projects may seek to keep the networks from different collaborations private from each other.
Searching and Filtering. Collaborators may search within and across these networks for proteins and interactions of interest. Collaborators may also want to examine subgraphs of a network.

Release. Upon the publication of a project, researchers may seek to make a subset of networks publicly available for access.

We address these challenges with GraphSpace, a web platform that satisfies these requirements. We believe that GraphSpace bridges the gap between systems like Cytoscape [189] that support visualization and analysis of individual networks and systems like NDex [151] that provide the network indexing capabilities. We anticipate GraphSpace to benefit interdisciplinary collaboration in network biology by allowing collaborating research teams to share, explore, and interpret networks and their drawings.

3.2 GraphSpace: System Description

We developed GraphSpace, a web-based system that addresses the challenges described earlier using a rich set of user-friendly features designed to enhance network-based collaboration (Figure 3.1). Users can upload richly-annotated networks, irrespective of the algorithms or software used to generate them. Users can create private groups, invite other users to join groups, and share networks with group members. A user may search for networks for specific attributes and for the presence of a specific node or nodes. A powerful layout editor allows users to efficiently modify node positions, edit node and edge styles, save new layouts, and share them with other users (Figure 3.9). Users may make networks public and provide a persistent URL in a publication, enabling others to explore these networks. Figures 3.2a–3.12 illustrate a user’s interaction with the GraphSpace web interface. While
users can view and interact with GraphSpace networks anonymously, several features become available after a user creates an account and logs in. We describe the main features of GraphSpace in this section. Section 3.3 and 3.4 discusses implementation details and use cases for GraphSpace respectively.

Figure 3.1: A typical GraphSpace workflow. A user may upload networks in the Cytoscape.js-supported JSON format into GraphSpace. The user and collaborators can explore networks using search, sharing, annotation, and filtering capabilities. Users may also create manual layouts of the networks (also shown in Figure 3.9A). Upon publication, networks may be shared with the scientific community via URLs. These networks may be used by others for their research, and re-imported into GraphSpace for further collaborations.
3.2.1 Sharing and Organization

Groups lie at the core of GraphSpace’s functionality. Any GraphSpace user may create a group and add other GraphSpace users as members of the group using their email address. Group owners can also invite other users by sharing a signup link for the group. Groups are private by design. A member of a group may share any network he or she owns with the group (“Sharing panel” in Figure 3.1). Only members of the group can view the shared network. Groups provide a level of privacy that is intermediate between the default (a network is visible only to its owner) and public networks (a network is visible to anyone; see “Public Networks” below).

3.2.2 Searching across Networks and within a Network

A group may contain hundreds or even thousands of networks. The search interface in GraphSpace allows a user to search for networks that have a specific attribute or tag and contain one or more nodes using simple syntax described below (Figure 3.2a). When the user visits a matching network, GraphSpace highlights the nodes that match the search term. An identical search interface on the page for an individual network enables the user to refine the query. This interface also enables the user to search for specific edges within the network (Figures 3.2b–3.3). GraphSpace allows the user to record the URL to the network with the search terms, for sharing or for further study.

**Search Syntax.** GraphSpace supports three types of search terms:

1. a single string, e.g., “wnt”. In this case, GraphSpace will return a network (a) if its “name” attribute contains the query as a substring or (b) if any node in the network has a “name”, “label”, or “aliases” attribute that contains the query as a substring.
2. two strings separated by a colon, e.g., “name:wnt”. Here, GraphSpace will return a network if the “data” section of its JSON representation contains an attribute called “name” whose value contains “wnt” as a substring.

3. two strings separated by two colons, e.g., “wnt::fzd”: This type of search term is only available when a user is searching a specific network. GraphSpace will highlight every edge that connects a node that matches “wnt” to a node that matches “fzd”. This search ignores the direction of the edge.

All searches are case-insensitive. A user may specify more than one search term. When the user searches all networks, GraphSpace returns only those networks that match all the search terms. When the user is visualizing an individual network and searching within it, GraphSpace highlights nodes and edges that match any search term (Figure 3.3).

![Figure 3.2: Screenshots of GraphSpace search functionality on the Graphs page. (a) The Graphs page, after searching for “ctnnb1” as a substring within the networks. (b) The user clicks on the network with the name “KEGG-Wnt-signaling-pathway” and reaches the network for the Wnt pathway with the searched node highlighted. This network is available at http://graphspace.org/graphs/user1@example.com/KEGG-Wnt-signaling-pathway.](image)
Figure 3.3: Screenshot of edge search functionality on the Graph page. The user can also add search terms for edges (“wnt::fzd” in this example). GraphSpace highlights any edge whose tail node matches “wnt” and whose head node matches “fzd”.

3.2.3 Creating, Saving and Sharing Layouts

GraphSpace exposes a subset of the layout algorithms available in Cytoscape.js [58] (the “Change Layout” button in Figure 3.9(B)). These algorithms often do not create layouts that match biological intuition (e.g., the spring embedder layout at the top right of Figure 3.1 and Figure 3.4). As an alternative, GraphSpace includes a powerful and intuitive layout editor that allows the user to quickly select nodes with specific properties (e.g., color and shape), to organize the selected nodes in defined shapes (e.g., a horizontal line or a filled circle), and to adjust the spacing between them (Figures 3.9(B)). In addition, the user can manually move nodes into any configuration (Figure 3.9(A)). The layout editor also allows users to manually change the visual properties of nodes and edges (Figures 3.5–3.8). Finally,
the user can save the layout, make this layout appear by default, and share it with other viewers of the network. Only users that have access (i.e., owned or shared via a group) can manipulate a network layout.

Figure 3.4: Screenshot of change layout panel on the Graph page. Clicking on the “Change Layout” button displays a layout panel which provides two alternatives. i) “Select Layout Algorithm”: This section lists all the automatic network layout algorithms supported by GraphSpace through its use of Cytoscape.js. ii) “Select Saved Layout”: This section lists layouts saved by the user. The user has created them in earlier sessions by using the layout editor or manually modifying the positions of nodes and edges created by some automatic layout algorithm and saving the layout. The user can click on the “Back” button to return to the main menu.
Figure 3.5: Screenshot of node editor panel before making a change in node(s) style. The figure displays the node editor panel when the user clicked on “Edit selected nodes” button after selecting the ‘CTNNB1’ node. The user may select more than one node. The options that appear on the right allow the user to change the following style properties of the selected node(s): i) Color: The background color of the node. ii) Shape: The shape of the node. iii) Width: The width of the node in pixels iv) Height: The height of the node in pixels v) Label: The label inside the node.
Figure 3.6: Screenshot of node editor panel after making a change in node(s) style. The figure displays the layout after the user makes the following changes: (i) color from green to pink, (ii) shape from rectangle to ellipse, (iii) width and height from 45px to 100px, and (iv) label from “CTNNB1” to “Updated CTNNB1”. At this point, the user can apply these changes by clicking on the “OK” button or discard them by clicking on the “Cancel” button. In the remaining figures in this section, we assume the user discards these changes and goes back to the initial layout.
Figure 3.7: Screenshot of edge editor panel before making a change in edge(s) style. Clicking on “Edit selected edges” button after selecting an edge opens the edge editor. The user may select more than one edge. The panel on the right allows the user to change the following style properties of the selected edge(s): i) Line Color: the colour of the edge, ii) Line Style: the style of the edge. iii) Width: the width of the edge in pixels, iv) Source Arrow Shape: the shape of the edge’s source arrow, and v) Target Arrow Shape: the shape of the edge’s target arrow.

Figure 3.8: Screenshot of node editor panel after making a change in edge(s) style. The layout after the user makes the following changes: (i) line color from grey to red (ii) line style from solid to dashed (iii) width from 3px to 10px iv) target arrow shape from none to triangle. At this point, the user can apply these changes by clicking on the “OK” button or discard the changes by clicking on the “Cancel” button.
3.2. GraphSpace: System Description

Figure 3.9: An example of layout editor workflow on GraphSpace. The graph used in this example is reconstruction of the Wnt pathway by the PathLinker algorithm [161] available at http://graphspace.org/graphs/?tags=graphspace-paper. (A) The layout editor provides an intuitive tool palette that allows the user to group nodes by colors and/or shapes and arrange them in blocks, circles, or lines to quickly build a custom-made manual layout of large networks. (B) The final layout of the network visualized in GraphSpace. The RYK-CFTR-DAB2 path referred to in the PathLinker use case in Section 3.4 appears on the left. The “Graph Information” tab displays a user-provided legend of shapes and colors. “Export” allows saving the visualization of a network in PNG and JPEG formats. This button also allows saving the structure and visual style of a network in JSON-formatted files.
3.2.4 Filtering Network Nodes and Edges

Graph algorithms may output networks where nodes and edges can be ordered, e.g., by path index [161] or by edge weight/score. GraphSpace allows each node and edge to have a integer-valued attribute called “k” that specifies the rank of the node or the edge (“Filter nodes and edges” panel in Figure 3.1 and Figures 3.10–3.12). For any network that contains this attribute, the user can adjust a slider in the panel to view only those nodes and edges with values of $k$ below a threshold. This interface element allows the user to unveil the network gradually in real time and gain intuition about how the network expands or contracts as this threshold changes.

Figure 3.10: Screenshot of filter panel on the Graph page. The figure displays the filter panel with “Current rank” slider when the user clicks on “Filter nodes and edges” button on the right hand side of the “Graph Visualization” tab. Moving the slider to the left removes all nodes and edges with the “k” attribute larger than the slider value without changing the positions of the remaining nodes and edges.
Figure 3.11: Screenshot of filter panel when only nodes and edges with “k” \( \leq 100 \) are visible.

Figure 3.12: Screenshot of filter panel when only nodes and edges with “k” \( \leq 10 \) are visible. When the user clicks on “Back” button, they return to the main menu in the “Graph Visualization” tab.
3.2.5 Tags

GraphSpace uses tags as a mechanism for grouping a set of graphs, e.g., all the graphs that a user may desire to make public upon the publication of a paper. A graph may have any number of tags. GraphSpace includes an interface to search for networks that match one or more tags. Tags, which organize graphs, provide a system that is orthogonal to groups, which organize collaborators.

3.2.6 Public Networks

An owner may make a specific network or all networks associated with a tag public, e.g., as accompanying information for a publication. All public networks and tags can be accessed using a URL to GraphSpace. For example, several automated reconstructions of human signaling pathways [161] appear at http://graphspace.org/graphs?tags=2015-npj-sysbio-appl-pathlinker.

3.2.7 Uploading Networks

GraphSpace networks follow the JavaScript Object Notation (JSON) format supported by Cytoscape.js.

Network format. GraphSpace networks follow a Cytoscape.js supported JSON format that separates the specification of the network structure (nodes and edges) from the description of the visual styles of the nodes and edges (e.g., colors, widths, shapes, labels). The same format can be exported from or imported to Cytoscape (version 3.1.1 or greater). GraphSpace treats some attributes specially:

- Cytoscape.js supports a network-level “data” section in the JSON file specifying the
3.2. **GraphSpace: System Description**

Figure 3.13: Screenshot of “Graph Information” tab on the Graph page. The “Graph Information” tab displays information about the network, e.g., a legend of node and edge shapes and colors. The “description” attribute in the JSON for the network specifies this content.

network structure. In this section, we allow users to specify attributes such as “name”, “title”, “description”, and “tags”. GraphSpace displays the “name” attribute to identify networks in the list that a user can access and in the list that match search results. When the user accesses a specific network, GraphSpace displays the “title” above the layout of the graph and the content of the “description” attribute in the tab called “Graph Information” (Figure 3.13) We also allow the “data” section to include network-specific attributes in this section, e.g., “PMID”, “authors”, and “organism”. GraphSpace displays all these additional attributes in “Graph Information” tab as well.

- Cytoscape.js also supports a “data” section within each node. Currently, GraphSpace recognizes an attribute called “aliases” in this section, which specifies a list of aliases for the node.
GraphSpace adds a “popup” attribute to each node and edge that allows a user to add HTML-formatted information, e.g., Gene Ontology annotations and database links for a protein; or types, mechanism, and database sources for an interaction (Figure 3.1).

Each node and edge may also have an integer-valued attribute called “k”, which denotes a rank. Through this attribute, GraphSpace allows the user to filter nodes and edges in a network visualization (Section 3.2.4).

Ways to upload networks. GraphSpace allows a user to upload a network in three different ways:

1. Users may create their own JSON files and upload networks one-by-one at http://graphspace.org/upload.

2. Cytoscape users may export their visually-coded networks from Cytoscape and import them directly into GraphSpace. This functionality works as follows. Since v3.1, Cytoscape has supported export of the structure of networks into JSON files and the visual elements of networks in JSON-based style files. GraphSpace can import both these types of files at http://graphspace.org/upload. Moreover, users can import a Cytoscape style file when they are editing the layout of a network in GraphSpace. In the future, we intend to develop a Cytoscape app to make the integration between the two systems seamless.

3. A comprehensive RESTful API and a Python module called “graphspace_python” facilitate bulk uploads of networks. Both the API and the module are easy to integrate into software pipelines.
3.2.8 Programmatic Access

RESTful APIs. The GraphSpace REST API provides endpoints for entities such as graphs, layouts, and groups that allow developers to interact with the GraphSpace website remotely by sending and receiving JSON objects. This API enables developers to create, read, and update GraphSpace content from client-side JavaScript or from applications written in any language. After a network is uploaded, the API allows the network owner to modify, delete, and share it. The API also allows a user to access several group management features available through the web interface. For example, a user can create or remove a group, add or remove members, obtain a list of groups he or she belongs to, and get information such as the membership on a group.

Python module. The GraphSpace software also includes a simple yet powerful Python library called “graphspace_python” that allows a user to rapidly construct a network, add nodes and edges, modify their visual styles, and then upload the network, all within tens of lines of code. Moreover, the user need not know the details of the REST API to use this module. It is very easy to integrate this library into a user’s software pipeline.

3.3 GraphSpace Implementation

GraphSpace uses Django (https://www.djangoproject.com), a popular, high-level Python web framework, for server-side functionality. It uses PostgreSQL for storing and querying its internal database of users, groups, graphs, nodes, edges, and layouts. GraphSpace also uses ElasticSearch as a NoSQL datastore for the JSON documents storing the network structure. This Apache Lucene-based indexing engine allows us to perform queries on the contents of the “data” attributes of networks and nodes. The distributed nature of ElasticSearch
also offers us the ability to easily scale our systems to accommodate increased bandwidth in the future. GraphSpace takes advantage of three open-source JavaScript libraries for its client-side functions: (i) Bootstrap (http://www.getbootstrap.com), an HTML, CSS, and JavaScript framework for the user interface; (ii) JQuery (http://www.jquery.com) for easy access to the Document Object Model (DOM), fast event handling, and Asynchronous JavaScript and XML (AJAX) calls; and (iii) Cytoscape.js [58] for network layout and visual styles for nodes and edges.

3.4 GraphSpace Use Cases

We provide two examples of how we have used GraphSpace in collaborations. First, we sought to determine regulatory mechanisms to extend a mathematical model of the budding yeast cell cycle [150]. For several query proteins, we computed the 50 shortest paths in a yeast interactome to the proteins in the model. We uploaded these networks to GraphSpace, annotating each node with the links to the Saccharomyces Genome Database and each edge with the source database and PubMed ID of the publication. These networks are available at http://graphspace.org/graphs/public?tags=2013-jcb-linker. Examination of the network for CDC5 (Polo kinase) revealed an unexpected connection between the proteins Bub2 and Tem1. The publication that reported this interaction provided evidence that allowed the modification of the mathematical model to correctly simulate the phenotype of yeast cells lacking Cdc5 and Bub2 [150].

Next, we designed an algorithm that reconstructs a human signaling pathway within an interactome when provided only the receptor and transcriptional regulator (TR) proteins in that pathway [161]. For over 20 pathways, we visualized the 200 shortest paths from any receptor to any transcriptional regulator using GraphSpace. The filtering panel on the reconstruction
of the Wnt signaling pathway revealed that the RYK-CFTR-DAB2 was the first receptor-to-
TF path that involved a protein (CFTR) not already known to be a member of the Wnt path-
way (left-most path in Figure 3.9(B)). Subsequent siRNA experiments in HEK290 cells con-
firmed CFTR’s role in canonical Wnt signaling and suggested the RYK-CFTR-DAB2 path as
an amplifier of Wnt signaling to β-catenin [161]. These pathway reconstructions are available

Since then, GraphSpace has proven useful for other projects involving remote collaborators
spanning multiple scientific disciplines [161, 167, 200] and in diverse contexts, such as ex-
ploring significant transcriptional links among biological processes [130], studying multi-way
molecular reactions [160], and exploring crowd-driven network layout methodologies [184].
GraphSpace has also been an invaluable tool for visually debugging initial implementations
of network biology algorithms. We have adopted GraphSpace for multiple undergraduate re-
search and capstone projects, demonstrating its ease of use and potential as a teaching tool.
Recently, several undergraduate students used the “graphspace_python” module to con-
struct and post networks to GraphSpace as part of a computational biology course at Reed
College. The majority of these students were biology majors with very little programming
experience.

3.5 Chapter Summary

The main contribution of this chapter is the GraphSpace system. It is a web-based platform
that fosters team science by allowing collaborating research groups to easily store, interact
with, layout and share networks. GraphSpace’s code is open source and available at http:
//github.com/Murali-group/graphspace if a user wants to run his or her own server. We
have hosted GraphSpace at http://graphspace.org. A more detailed documentation and
tutorial on GraphSpace can be found at http://manual.graphspace.org/.
Chapter 4

Flud: a hybrid crowd-algorithm approach for visualizing biological networks

This chapter is based on a manuscript whose revision is under review in the journal ACM Transactions on Computer–Human Interaction (ToCHI).

Aditya Bharadwaj, David Gwizdala, Yoonjin Kim, Kurt Luther, T M Murali; “Flud: a hybrid crowd-algorithm approach for visualizing biological networks.” Major revision under review at ACM Transactions on Computer–Human Interaction (ToCHI).

4.1 Motivation

Modern experiments in many disciplines generate large quantities of graph data. Chapter 3 presents GraphSpace, a solution that allows users to store, organize, share, and explore their graph data on a single online platform. GraphSpace also allows researchers to disseminate and analyze, by manually creating layouts that clearly convey the domain knowledge underlying the network data. However, the graph layout problem remains challenging due to multiple conflicting aesthetic criteria and complex domain-specific constraints.
In this chapter, we present explore and present a hybrid crowd-algorithm strategy for balancing multiple conflicting aesthetic criteria and a complex domain-specific constraint for a biological graph visualization task. Our proposed strategy can help network biologists understand the protein interactions that underlie processes that take place in the cell. We facilitate this strategy with a crowd-powered system, Flud, wherein crowd workers and a simulated annealing algorithm build on each other’s progress. Flud also provides a game interface that allows humans with no expertise to design biologically meaningful graph layouts with the help of algorithmically generated suggestions. We use GraphSpace as an backend infrastructure for storing graphs and their drawings created by Flud and later making them accessible for exploration and analysis.

We experimented with three different hybrid crowd-algorithm approaches, a crowd-only approach, and state-of-the-art techniques to understand the impact of mixed-initiative approach on the quality of generated layouts. We also conducted user-study with participants with biological training to showcase the task effectiveness of Flud layouts compared to those created by state-of-the-art techniques. We also studied the impact of algorithmically generated suggestions on the quality crowd work.

4.2 Description of the Flud System

Flud is a crowd-powered system that allows a requester — a biologist seeking to visualize his or her network data — to crowdsource the layout design task for biological network visualizations to novice crowd workers. It has two main components: the requester interface and the worker interface. We now describe how Flud assists novice crowd workers in visualizing and laying out a network in the context of layout criteria specified by a requester.
4.2. Description of the Flud System

4.2.1 Requester Interface

This interface allows a requester to send a network to Flud and crowdsource the layout design task. The requester can use the interface to control parameters such as the number of crowd workers per design task, layout criteria, and the amount of time a crowd worker can spend on the layout design task. We detail some of the most important parameters below.

Layout Criteria

A requester can ask Flud crowd workers to optimize the network layout for five types of criteria (see Figure 4.1). The first criterion is domain-inspired, while the other four are aesthetic and have been previously used as design guidelines in network drawing (also called graph drawing).

1. *Downward pointing paths*: This guideline asks crowd workers to maximize the number of downward pointing paths from a given set of source nodes (e.g., receptors) to target nodes (e.g., transcription factors). These types of paths draw visual attention to sequences of edges that lead from receptor proteins in the cell membrane (green triangles, placed at the top of the layout) through internal nodes to effector molecules in the nucleus (transcription factors, yellow squares, placed at the bottom of the layout). This domain-specific constraint is especially useful for analyzing the flow of information in biological networks that represent cellular signaling.

2. *Non-crossing edge pairs* [155, 196, 199]: The goal of this guideline is to maximize the number of edge pairs that do not intersect, i.e., minimize the number of edge crossings.

3. *Edge length* [196, 199]: This guideline asks crowd workers to minimize the length of the edges in the network.
Figure 4.1: Five layout criteria used in Flud to allow effective interpretation of signaling flow from source nodes (green triangles, placed at the top of the layout) to target nodes (yellow squares, placed at the bottom of the layout). 1) Minimize the number of edge crossings, 2) Keep nodes connected by an edge close to each other, 3) Disperse disconnected node pairs, 4) Increase the separation between nodes and edges, and 5) Increase the number of downward pointing paths.

4. **Node distribution** [18, 38, 196, 199]: In this guideline, unconnected nodes should be far apart from each other.

5. **Node edge separation** [38]: This guideline seeks to create layouts where nodes are positioned away from edges.

Requesters may also assign priorities or weights to each layout criterion to convey their relative importance to Flud crowd workers. These priorities help crowd workers to prioritize layout criteria in case of conflicts. A requester can also exclude a layout criterion by assigning it a priority of zero.
Crowdsourcing Approach

One of the challenges in creating network layouts is that different criteria may conflict with each other. Heuristics used by automated methods may compute non-optimal solutions or get stuck in local optima. For instance, the correct orientation of several edges in a path may be required to make it point downwards. Similarly, several edges may have to manipulated to remove an edge crossing without introducing new ones. In another example, the length of multiple edges may have to be adjusted to increase the distance between a connected pair of nodes in a path. Automated methods, in general, find it hard to perform such complex improvements to the layout while balancing all the criteria. Therefore, it is a common practice for experts (biologists in our case) to manually improve automatically generated visualizations. In Flud, we use crowdsourcing to leverage the visual and cognitive abilities of humans to observe patterns and identify solutions that escape local optima.

Flud allows requesters to specify the total number of crowd workers for a layout design task. They can also select one of two available crowdsourcing approaches:

1. **Crowd**: In this approach, Flud asks a fixed number of crowd workers (specified by the requestor) to lay out a graph one after another. Each crowd worker starts their session with the highest-scoring layout created so far, i.e., across all earlier sessions. During a session, a worker may create multiple layouts. Flud stores the highest-scoring layout of these. If this layout’s score is better than the current leader, Flud updates the best overall layout. In this fashion, crowd workers can iteratively improve upon one another’s results.

2. **Hybrid**: Flud alternates sessions between crowd workers and a simulated annealing-based layout algorithm [38]. In each session, either a crowd worker or simulated annealing starts with the highest-scoring layout created so far, with this layout updated as in
the crowd-only approach. In this fashion, crowd workers and the simulated annealing algorithm can iteratively improve upon one another’s results. Flud allows requesters to specify the initial temperature (default = 100) and number of iterations (default = 500) in a given simulated annealing session.

### 4.2.2 Worker Interface

The worker interface has three major parts: a visualization of the network being laid out on the left, text panels with task-related tips in the middle, and a sidebar with layout controls and score information on the right (Figure 4.2).

![Figure 4.2: Screenshot of Flud’s worker interface in the “Downward Pointing Paths” mode. The interface has three major parts: a visualization of the network being laid out (left), text panels with task-related tips (middle), and a sidebar with layout controls and score information (right). The upward pointing edges are shown in red to draw the crowd worker’s attention. The path highlighted in saturated colors is an automatically generated hint. Reorienting the red edge to make it point downward will increase the number of downward pointing paths.](image-url)
4.2. Description of the Flud System

Network Visualization

Flud provides an interactive network visualization that supports both touch and mouse gestures to select and drag one or more nodes. Since Flud crowd workers do not need biological expertise, they do not see any node or any edge labels describing their biological meaning. Instead, each node displays the number of edges incident on it as a signal of that node’s importance.

Worker’s reward

Flud incentivizes workers to improve the layout by granting them a bonus pay based on how much they improve it [100, 129]. To measure the improvement, Flud uses a layout scoring system that considers the specified layout criteria and their priority values. The top portion of the sidebar displays in real-time the highest score achieved by a layout for the network, the score of the current layout, and improvement in score. Flud also synchronously converts the improvement in score to a bonus payment for workers and displays it in a progress bar below these scores.

Criterion-specific Scoreboard

The scoreboard displays the scores and priorities (in coin-shaped badges) for each individual criterion. In the scoreboard, we sort the criteria in decreasing order of the priorities assigned by the requester. We scale the per-criterion scores between 0 and 10,000 to avoid displaying floating point values. After the crowd worker makes a move, Flud recalculates and displays, in real time, each per-criterion score as well as the total score. A green up arrow (↑) or a red down arrow (↓) next to each per-criterion score allows crowd workers to track the impact of their last move on the score. In addition, we display the change in the overall score with a
Chapter 4. Flud: a hybrid crowd-algorithm approach

similar color scheme.

Criterion-specific Nodes and Clues

Csikszentmihalyi [37] argues that a user can achieve flow, a state of being “in the zone,” if they attain multiple component states. These states include challenge-skill level balance, immediate and unambiguous feedback from the system, concentration on the task at hand, and clarity of goals. Inspired by flow theory, which has been influential in game design [34], we implemented two important features — criterion-specific modes and clues (Figures 4.5 and 4.6) — to help crowd workers focus on their goals and make progress without becoming bored or frustrated. At the start of a session, Flud assigns the crowd worker one of the layout criteria as a criterion-specific mode. In a mode, to delineate the task, the visual representation of the network highlights (in red) the elements that are relevant to the criterion-specific task. Flud has five such modes: Downward pointing paths, Non-crossing edge pairs, Edge length, Node distribution, and Node edge separation. Flud allows requesters to specify the order in which criterion-specific modes are assigned in a sequence. However, irrespective of the assigned mode, the crowd worker’s overall goal is to create a layout that optimizes the total score based on all the criteria (see Section 4.2.1).

Moreover, if the crowd worker is stuck during the design task, Flud reduces the challenge level by presenting them with an algorithmically-generated, mode-specific “clue” that highlights a small subset of nodes and edges in the network and further narrows the focus of a crowd worker to a very specific task [214]. Flud selects specific elements in the clue such that changing their positions is likely to improve the score for the corresponding criterion. We now describe each criterion-specific mode and the corresponding clue.
(i) **Downward pointing paths:** We say that an individual directed edge is *downward pointing* if the $y$-coordinate of its head is below the $y$-coordinate of the tail and angle between the edge and the $x$-axis is greater than equal to a fixed degree (15 degrees, in our implementation; Figure 4.3). A path is *downward pointing* if all the edges in it have this property (Figure 4.4).

![Figure 4.3: Illustration of downward and upward pointing edges. A directed edge is downward pointing if the $y$-coordinate of its head is below the $y$-coordinate of the tail and angle between the edge and the $x$-axis is greater than equal to 15 degrees. The blue and red regions show the angles at which a directed edge will be considered as downward pointing and upward pointing, respectively.](image)

![Figure 4.4: Illustration of downward-pointing paths. (a) Three paths. (b) Moving the node with a green border increases the number of paths to four. (c) Another move increases the number to six.](image)

In the downward pointing paths mode, Flud highlights every upward pointing edge in red.
Chapter 4. Flud: a hybrid crowd-algorithm approach

(Figure 4.5a). A crowd worker can reorient such an edge in an attempt to increase the number of downward pointing paths. A clue in this mode (Figure 4.5b) highlights a path in the network that (a) contains at least one upward pointing edge, and (b) has the property that reorienting all upward-pointing edges in the path is guaranteed to increase the number of downward pointing paths. If there is no such path, the crowd worker does not see any clue.

(ii) Non-crossing edge pairs: This mode displays every edge crossing as a red point (Figure 4.5c). When a crowd worker hovers on an edge crossing, the mode highlights the intersecting edges. Decreasing the number of edge crossings is perhaps one of the most challenging layout criteria since the crowd worker has to manipulate nodes without introducing new intersections. Intuitively, the higher the degree of a node, the harder it may be to remove intersections involving edges incident on that node. Therefore, the clue in this mode highlights a pair of crossing edges such that the total degree of the four nodes involved is the smallest over all the intersections in the layout (Figure 4.5d). Note that we cannot guarantee that the crowd worker will indeed be able to decrease the number of crossings by moving one or more of these nodes.

(iii) Edge length: For this criterion, crowd workers need to consider the length of all the edges in the network. Therefore, the mode presents all the nodes and edges in blue. The clue highlights an edge that is either very long or very short (Figure 4.6a). Correcting the length of this edge should lead to the improvement of the edge length score.

(iv) Node distribution: In this mode, the clue highlights the unconnected pair of nodes that are the closest to each other (Figure 4.6b). Thus, the clue provides a greedy way for crowd workers to consider node pairs that need to be moved further apart.
4.2. Description of the Flud System

(a) Mode for downward pointing paths.  
(b) Clue for downward pointing paths.

(c) Mode for edge crossings.  
(d) Clue for edge crossings.

Figure 4.5: Screenshots of the network visualization and the clue in the “Downward pointing paths” and “Non-crossing edge pairs” modes.
(a) Clue for edge length.
(b) Clue for node distribution.
(c) Clue for node edge separation.

Figure 4.6: Screenshots of the network visualization in Edge length, Node distribution, and Node edge separation mode.

(v) Node edge separation: Here, the clue highlights the node-edge pair that is the closest to each other (Figure 4.6c).

Layout Controls

The bottom part of the sidebar contains multiple controls that allow crowd workers to rapidly perform non-trivial changes to the layout. These changes include expanding and squeezing the spacing between selected nodes, undoing and redoing earlier actions, and reverting to the layout with the best score.

Tips

The middle part of the interface contains text panels with task related tips. These tips include an example of a good layout for a toy network, a link to the tutorial on how to use criterion-specific clues, and specific instructions on how to improve the per-criterion score.
4.2. Description of the Flud System

for the assigned mode.

Bounding Box

To prevent crowd workers from creating pathological layouts that move nodes indiscriminately far apart from each other, we set the scores for each criterion to zero if any node is moved outside a bounding box of fixed size.

4.2.3 Layout Scoring System

In describing the layout scoring system, we use $G = (V, E)$ to denote the network $G$ being laid out with $V$ denoting its nodes and $E$ denoting its edges. We use $n$ to denote the number of nodes and $m$ for the number of edges in $G$. Given the $x$- and $y$-coordinates for every node in $V$, we use $d(u, v)$ to denote the Euclidean distance between nodes $u$ and $v$ and $l(e)$ to denote the length of an edge $e$ (the Euclidean distance between its endpoints). We use $w$ and $h$ to represent the fixed width and height of the bounding box. We now describe how we computed the score corresponding to each criterion (per-criterion score).

1. **Downward pointing paths**: We compute a normalized downward pointing score as follows:

$$DP(G) = \frac{\pi(G)}{\rho(G)}.$$

where $\pi(v)$ is the number of downward-pointing paths and $\rho(v)$ is the approximated upper bound of the number of downward-pointing paths. The closer $DP(G)$ is to one, the larger the number of downward-pointing paths in the layout.

This criterion alone can be optimized using a topological sorting algorithm in directed acyclic graphs. However, biological networks often contain many feedback loops or
cycles, making it increasingly challenging to optimize this criterion as some of the edges have to left upward-pointing. Laying out a directed graph with cycles by leaving a few edges as upward-pointing to maximize the downward pointing paths is similar to the feedback arc set problem which is NP-hard [108].

2. Non-crossing edge pairs: Since the maximum number of non-crossing edge pairs possible in $G$ is $m(m - 1)/2$, we compute the non-crossing edge pairs score as

$$EC(G) = \frac{2\chi(G)}{m(m - 1)}.$$ 

The closer $EC(G)$ is to one, the smaller is the number of edge crossings in the layout. We have intentionally defined $EC(G)$ as above so that high values reflect good performance and crowd workers focus on increasing their scores. We also note that minimizing the number of edge crossings alone is an NP-complete problem [69].

3. Edge length: We define the cost $c(e)$ of an edge to be equal to its length $l(e)$ if $l(e) \geq$ size of a node or equal to a large penalty, otherwise. We normalize the cost of each edge by the largest possible edge length and therefore, penalty should be greater the diagonal of the screen. We compute the edge length score of a layout as

$$EL(G) = \max \left\{ 0, 1 - \frac{1}{m} \sum_{e \in E} \frac{c(e)}{\sqrt{w^2 + h^2}} \right\}$$

Thus, the closer $EL(G)$ is to one, the closer the nodes connected by an edge are, on average.

4. Node distribution: Here, we want to maximize the average distance in the layout between a node and its closest unconnected node. Unlike the previous criterion, it applies only to pairs of nodes not connected by an edge. We compute the node distribution
4.2. Description of the Flud System

score as

\[ ND(G) = \frac{1}{n} \sum_{u \in V} \min_{v \in V, (u,v) \notin E} \frac{d(u, v)}{\sqrt{w^2 + h^2}} \]

5. Node edge separation: In this criterion, we want to maximize the average distance between a node and the closest edge to it in the layout. Therefore, we define the node edge separation score as

\[ NED(G) = \frac{1}{n} \sum_{u \in V} \min_{e=\{t,v\} \in E, u \neq t \neq v} \frac{d(u, e)}{\sqrt{w^2 + h^2}} \]

where \( d(v, e) \) denotes the distance between node \( v \) and edge \( e \).

We define the overall score \( OS(G) \) for a layout as a weighted sum of the five per criterion scores

\[ OS(G) = w_{DP} \times DP(G) + w_{EC} \times EC(G) + w_{EL} \times EL(G) + w_{ND} \times ND(G) + w_{NED} \times NED(G), \]

where the coefficients are the priorities assigned to each criterion by the requester. We describe the algorithm and parameters used to compute the per criterion scores in Section A.2 in Appendix.

4.2.4 Recruiting and Training Crowd Workers

Since we do not require the workers to have network or biology expertise, Flud recruits novice crowd workers directly from the Amazon Mechanical Turk (MTurk) platform. As mentioned earlier, Flud allows the requester to configure the number of workers required for a network. In order to educate the workers to use the layout interface, we ask the crowd worker to go
through a two-part interactive tutorial that introduces the visual elements, task instructions, and use of the criterion specific clues. In the first part of the tutorial, we train the worker to improve the criterion-specific subscore both with and without the help of a clue. At the end of part one of the tutorial, we give the worker an option to submit the HIT or to continue with part two of the tutorial and lay out a network to earn a bonus (Refer to Section 4.3.4). The second part of the tutorial introduces the worker to the remaining subscores and the complete instructions for the layout design task.

In a pilot study where we asked crowd workers to perform the layout design task, we noticed that some workers (a) did not carefully go through the tutorial and (b) moved the nodes aimlessly to use up the fixed number of moves required to be paid for completing the layout design task. In order to prevent such unproductive crowd activity, we only invite crowd workers to do the design task if they correctly solve two simple puzzles towards the end of the first tutorial. The first puzzle asks the crowd worker to increase one of the per-criterion scores in a toy network by at least one point with the help of a clue. The second puzzle again asks the worker to increase the per criterion score for the same network except that there is no clue available. Informally, we observed that workers who solved these puzzles were later more engaged and likelier to improve the layout in the actual design task.

### 4.2.5 Simulated Annealing

We use Simulated Annealing [38] as the primary baseline for comparison and as the algorithm in the hybrid approach. We selected this method primarily because our goal is to find the global optimum and our objective function i.e., the overall layout score (Section 4.2.1) is non-differentialble in nature. Simulated Annealing satisfies both of these requirements and therefore gives us the flexibility to accommodate all of our criteria (Section 4.2.1). We run
4.2. DESCRIPTION OF THE FLUD SYSTEM

the algorithm for a fixed number of iterations (e.g., 500) with the temperature decreasing by a factor of 0.995 after every iteration. Each iteration involves $10n$ steps, where $n$ is the number of nodes in the network. In every step, we move a node to a random position $(x_{new}, y_{new})$ computed as follows from the current position $(x_{current}, y_{current})$:

$$
x_{new} = x_{current} + \rho_x \frac{1}{4} \left( \frac{T}{T_{max}} \right)^2 w
$$

$$
y_{new} = y_{current} + \rho_y \frac{1}{4} \left( \frac{T}{T_{max}} \right)^2 h
$$

where $T_{max} = 100$ and $T$ represent the maximum and current temperatures, respectively, and we draw $\rho_x$ and $\rho_y$ uniformly at random from the interval $[0, 1]$. Thus, we select the new position uniformly at random from a rectangle centered at the current location of the node. The size of this rectangle is proportional to the fixed page size and decreases quadratically with the temperature. We compute the score of the layout with this new position. We accept the move if it improves the score. Otherwise, we accept this move, even though it worsens the score, with a probability $Pr(\Delta s) = e^{-\Delta s / T}$, where $\Delta s$ is the change in score. Therefore, as the temperature decreases, so does the probability of accepting a move that worsens the score. Once the algorithm stops, we use the best layout in the entire run as the final layout.

Overall, the annealing schedule took approximately 1 hour to cool down from high initial temperature ($T = T_0 = 100$) to low temperature ($T \approx 1$). Flud allows requesters to specify the initial temperature (default = 100) and number of iterations (default = 500) in a given simulated annealing session. We also used Simulated Annealing to fine-tune [87] the final layout generated by the crowd workers. We describe the fine-tuning procedure in detail in Section A.3 in Appendix.
4.2.6 Implementation Details

We implemented Flud in Python using the Django web framework, with network visualization supported by Cytoscape.js \cite{59}. Flud interfaces with GraphSpace \cite{20} for storing and sharing networks and their layouts. The requesters can access all of the crowd-generated layouts, including the best overall layout on GraphSpace.

4.3 Evaluation

Having implemented the Flud system as described above, we aimed to showcase its effectiveness to crowdsource the layout design task for biological network visualizations. To this end, we posed the following research questions:

Q1: How should Flud assign criterion-specific modes to crowd workers to optimize the layout scores they achieve?

Q2: How do the layout scores achieved by mixed-initiative or crowdsourcing approaches in Flud compare to automated methods?

Q3: Can users with biological training more effectively understand Flud layouts compared to those created by automated methods?

In the rest of this section, we describe how we selected crowd workers, the networks we laid out, the layout design task each worker had to perform, and how we compensated them before presenting our experiment design.
4.3.1 Networks

We selected three different complex protein networks of similar size (Table 4.1) that represent signaling pathways in cells. These networks contained both directed and undirected edges. These networks contained a small (G1), medium (G2), and high (G3) number of cycles. The presence of a large number of cycles makes it difficult to create layouts with many downward-pointing paths. Hence, these networks should present different levels of challenges to crowd workers and to automated algorithms.

<table>
<thead>
<tr>
<th>Network</th>
<th>Description</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>#Simple Cycles</th>
<th>#Simple Paths from sources to targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Crosstalk from Estrogen signaling pathway to HIF-1 signaling pathway</td>
<td>71</td>
<td>112</td>
<td>3</td>
<td>1952</td>
</tr>
<tr>
<td>G2</td>
<td>Network representing Epithelial-to-mesenchymal transition (EMT)</td>
<td>69</td>
<td>131</td>
<td>8463</td>
<td>10066</td>
</tr>
<tr>
<td>G3</td>
<td>Signaling from SCH9 to TPK2 in budding yeast</td>
<td>58</td>
<td>136</td>
<td>269437</td>
<td>2361</td>
</tr>
</tbody>
</table>

Table 4.1: Networks used for evaluation. Refer to Section A.1 in Appendix for information on data availability.

4.3.2 Crowd Workers

In our experiments, crowd was recruited from the Amazon Mechanical Turk (MTurk) platform. We used MTurk’s built-in qualification types to only recruit workers from the US with a Human Intelligence Task (HIT) approval rate of at least 97% and at least 100 completed HITs.

4.3.3 Crowd Worker Task

At the start, we randomly select a task corresponding to one of the three networks. Next, we assign them one of the layout criteria (using one of the strategies described in Section 4.3.5) to restrict their task to a criterion-specific mode. Next, we ask the crowd worker to go
through a two-part interactive tutorial before starting the layout session. During the layout session, each worker had one hour to generate an improved layout. A worker can quit the task at any point.

4.3.4 Compensation

For the first half of the tutorial, we compensate the workers with at least the minimum hourly wage rate ($7.25 per hour) in our region. If a worker elects to continue to the layout design task in order to obtain bonus compensation, we pay this amount according to how they increase the score of the layout (refer to Section A.4 in Appendix for details).

4.3.5 Experiment Design

Given the central importance of domain-specific downward pointing path criterion, we assigned it a high priority of 400. We assigned the non-crossing edge pairs criterion a priority of 3 and rest of the criteria a priority of 1. We conducted three experiments to address our research questions.

Experiment 1

We designed the first experiment to answer Q1, where we want to find a good strategy to assign a criterion-specific mode to a crowd worker in a sequence. To this end, we evaluated the performance of two different approaches: Crowd and Crowd-Random. In the Crowd approach, we assigned criterion-specific modes to workers in the order of their priorities, i.e., downward pointing paths (DP), non-crossing edge pairs (EC), edge length (EL), node distribution (ND), and node-edge separation (NED), as shown in Figure 4.7.
4.3. Evaluation

We recruited 20 crowd workers for each task sequence where four \((N = 4)\) workers focused on each of the five layout criteria used in Flud. In contrast, for the Crowd-Random approach, we randomly assigned each criterion four \((N = 4)\) times in a sequence of 20 crowd workers (refer to Figure 4.7). We recruited crowd workers for three task sequences per network for each approach. Overall, we recruited 360 crowd workers for this experiment.

Experiment 2

In our second experiment, we evaluated the performance of the Crowd approach described in Experiment 1 and the Hybrid (Crowd-SA) approach against automated baseline methods with the goal of answering Q2. Similar to the Crowd approach, for Hybrid approach, we recruited
5N crowd workers for each task sequence where N workers focused on each of the five criteria; we provide the values of N below. However, in Hybrid, we alternate sessions between crowd worker and simulated annealing (refer to Figure 4.7). Davidson and Harel’s simulated annealing based layout algorithm [38] starts with “non-local” moves ($T = 100$), where a node’s next location is not restricted to nearby positions, and ends with “local” moves ($T \approx 1$) where a node can move only to a nearby position. In this experiment, we tried three different types of hybrid approaches (Crowd-SA100, Crowd-SA50, and Crowd-SA20), where we alternate the Crowd approach and simulated annealing schedules with three different initial temperatures (refer to Figure 4.8).

- **SA100 (High temperature).** In this schedule, we started with a high temperature ($T_0 = 100$) and only ran the initial part of the schedule (approximately 15 minutes). In this schedule, we only made random moves that are “non-local” in nature.

- **SA20 (Low temperature).** Here, we started with a low temperature ($T_0 = 20$) and only ran the later part of the schedule (approximately 15 minutes). Due to the low initial temperature, we only made random moves that are “local” in nature. Such moves further optimized the score while preserving the overall structure of the layout.

- **SA50 (Medium temperature).** In this variant, we started with an intermediate temperature ($T_0 = 50$) and ran the middle part of a SA schedule (approximately 15 minutes).

![SA schedule diagram](image)

*Figure 4.8: Figure illustrating the three simulated annealing schedules used in our hybrid approaches.*
Overall, we aimed to recruit $N = 10$, $N = 15$, and $N = 20$ crowd workers per criterion for networks G1, G2, and G3, respectively, for both Hybrid and Crowd approach. We stopped recruiting crowd workers once the total time taken by the workers in the sequence exceeded 24 hours. At the end of the sequence, we fine-tuned the best layout so far to generate the final output layout by using the fine-tuning procedure described in Harel and Sardas’s work on incremental improvement of layouts of planar networks.

In order to highlight the quality of layouts generated by the Flud system, we compared our results against four automatic layout algorithms.

- **Simulated annealing (SA)** [38]. We selected this algorithm as our primary baseline due to its flexibility to accommodate all of our criteria (Section 4.2.1). We ran the algorithm for a fixed number of iterations (e.g., 500) where the temperature cools down from high initial temperature ($T = T_0 = 100$) to low temperature ($T \approx 1$). To compare performance of SA against Crowd and Hybrid, we ran the simulated annealing schedule multiple times for 24 hours such that each schedule builds upon the best layout created so far. We ran the baseline simulated annealing algorithm on a machine with an Intel Xeon (16 cores at 2.40 GHz) CPU.

- **Dig-Cola** [46]. Our second baseline was Dig-Cola, which may indirectly optimize the number of downward pointing paths criterion since it lays out nodes in a hierarchical manner. Dig-cola also provides a way to conserve aesthetic criteria such as edge lengths and symmetries. We used the implementation of Dig-Cola in the Neato program in the Graphviz package [54]. We ran Dig-Cola with nine different values of the minimum gap between levels \{0.1, 0.2, 0.5, 1, 2, 5, −0.5, −1, −2\}, four different values of the preferred edge length parameter \{1, 3.125, 5\}, and all possible values of the overlap parameter. Finally, we used the layout that produced the largest total score.
• *IPSEP-Cola* [48]. Our third baseline was IPSEP-Cola, which tries to ensure that node $s$ is placed above node $t$ if there is a directed edge from $s$ to $t$. If the network contains directed cycles, IPSEP-Cola computes the largest acyclic subgraph and relaxes the constraints on the excluded edges. This approach may optimize the number of downward pointing paths. We used the implementation of IPSEP-Cola in the Neato program in the Graphviz package [48]. We ran a parameter search similar to Dig-Cola and used the layout that produced the largest total score in our comparisons.

• *Spring-electrical model* [98]. This force-directed algorithm uses spring elasticity to keep connected nodes closer and electric charge repulsion to keep disconnected nodes away from each other. The method is available as the ‘sfdp’ program in the Graphviz package [54]. We ran the program with six different values of the spring constant $K = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$ and of the power of repulsive force $R = \{0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 30, 40\}$. We used the layout from the parameter pair with the highest score in our comparisons.

• *CerebralWeb* [60]. While the other automated baseline methods do not use any biological information, CerebralWeb uses the cellular localization annotations of proteins to determine node location. Specifically, CerebralWeb constrains the nodes in layers like “plasma membrane”, “cytoplasm”, and “nucleus”. CerebralWeb orders these layers in the layout such that it resembles the way biologists lay out signaling pathways in textbooks and publications. We used the CerebralWeb Javascript plugin for Cytoscape.js to generate layouts. We ran CerebralWeb three times and selected the highest scoring layout for our comparisons.

Finally, we also instrumented Flud to record each action taken by the crowd workers during their layout design task as well as the corresponding scores and layout. We used the collected data to analyze how the crowd workers optimized the scores, and understand the impact of
features such as criterion-specific modes and clues on the scores.

Experiment 3

In the third experiment, we conducted a user study to compare the quality of layouts with the goal of answering Q3. To this end, we evaluated a layout’s effectiveness [164] in supporting human understanding of signaling information represented in the biological networks. We selected a task that is common for scientists in network biology and fundamental to understanding the signaling mechanism — interpreting how “causes” (e.g., receptors) are linked to “effects” (e.g., transcription factors) in the underlying networks. Specifically, we asked participants to sort a given set of nodes in the order in which signaling information appears to flow through them from a source node to a target node. Such a task requires the participants to correctly identify the connections between the given nodes and how they constitute a source-to-target path.

We restricted the time spent by the participants on each task due to two considerations. First, a common goal of data visualization is to support fast comprehension. Second, in a pilot study, we noticed that participants frequently spent a lot of time on poor layouts before submitting a solution or simply giving up the task. We also found that the participants required less than 60 seconds on average to perform these tasks successfully. With these points in mind, we set a 60 second time limit on each task. In the pilot study, we also found that the participants spent a considerable amount of time in searching for the given nodes in the layout, which is not the primary goal of the task. Therefore, to ensure that the participants can quickly start the task, we highlighted the given nodes in the layout. Participants were also allowed to pan and zoom to different parts of the layout.

For the main experiment, we recruited seven graduate students (five doctoral, two master’s)
Chapter 4. Flud: a hybrid crowd-algorithm approach

Figure 4.9: Screenshots of the interface used in task-based effectiveness evaluation described in Experiment 3. Each task has three questions: (A) Question 1 asks the participants to sort the nodes in the order in which information appears to flow from source node to target node. (B) Question 2 and 3 asks the participant to rate their confidence in the submitted answer and easiness of the task on a five-point Likert scale.

with a research background in computational biology as participants (P1–P7). We generated four tasks for each graph by randomly selecting paths of length 9, 10, 11, and 12. We selected the best layouts generated by Flud and each of the automated baseline methods in Experiment 2. Thus, each participant was assigned 60 tasks (3 graphs × 4 paths × 5 layouts). To mitigate learning effects, we relabeled the nodes in the layout for each task and also presented the tasks in random order. Each task contained three questions. The first question asked participants to sort the nodes in the order in which information flows within 60 seconds. Figure 4.9A displays screenshots of the question page. Next, participants were asked to rate their confidence in their answer and ease in completing the task on a five-point Likert scale. These two questions did not have a time limit. Screenshots of these questions are shown in Figure 4.9B. We also instrumented the experiment to record pan and zoom user interactions and analyzed the logs, as described in Section 4.4.3.

Finally, we also conducted semi-structured interviews in person with each participant after
they completed the task. All interviews were audio-recorded and partially transcribed based on detailed notes. We used a bottom-up approach to analyze these transcripts and organized the findings around the three questions in the task.
4.4 Results

4.4.1 Q1: How should Flud assign criterion-specific modes to crowd workers to optimize the scores they achieve?

![Figure 4.10: Comparison of scores achieved by crowd workers when modes are assigned randomly (Crowd-Random) and in decreasing order of priorities (Crowd). (A) Distributions of total scores. (B) Distributions of per-criterion scores.](image)

We used Experiment 1 to compare the performance of the crowdsourcing approach when the criterion-specific modes are assigned in decreasing order of the priorities (i.e., Crowd) versus
4.4. Results

A random assignment of modes (i.e., Crowd-Random). Figure 4.10A shows that for each of the three networks, the median total score achieved by the Crowd approach was greater than the median score achieved by Crowd-Random approach. On further investigation, we found out that the Crowd approach achieved a better median per-criterion score for downward pointing paths (DP), node distribution (ND), and node edge separation (NED) criterion for all three networks (Figure 4.10B). On the other hand, the Crowd-Random approach achieved a higher median per-criterion score for non-crossing edge pairs (EC) and edge length (EL) criterion for networks G2 and G3. Since the Crowd approach achieves a higher score for three out of five layout criteria, including the important domain-specific DP criterion (reflected in the total layout score), we decided to assign criterion-specific modes in order of priorities in Experiment 2.

In the Crowd approach, we also observed that while four crowd workers were able to lay out as many as 1,303 downward pointing paths in network G1, the same number of crowd workers could only lay out 321 and 131 downward pointing paths in networks G2 and G3, respectively. We attributed this difference in performance to a large number of cycles in G2 and G3 compared to G1 (Table 4.1). We noticed that in the presence of large numbers of cycles, crowd workers had to make more moves in network G2 (mean=899) and G3 (mean=850) compared to network G1 (mean=403) where the number of cycles is very low (Table 4.1). Therefore, we decided to recruit crowd workers in proportion to the number of simple paths from sources to targets (Table 4.1) in Experiment 2 to balance out the crowd workers’ low throughput for downward pointing paths criterion in networks with large number of cycles.
4.4.2 Q2: How do the layout scores achieved by hybrid or crowd-sourcing approaches in Flud compare to automated methods?

We used the data collected from Experiment 2 to answer this research question. We considered several aspects to answer RQ2: overall scores, number of downward-pointing paths, scores for aesthetic criteria, and rate of improvement.

![Figure 4.11: Distributions of the total scores of layouts created by different approaches.](image)

*Overall score.* First, we compared the performance of all the approaches on the overall score (Figure 4.11). We observed that the crowd and hybrid approaches clearly outperformed all the automated techniques for networks G2 and G3. In contrast, for network G1, the automated methods, especially SA, had comparable performance to the crowd and hybrid approaches. We also noticed that *Dig-Cola*, *IPSEP-Cola*, and *Spring Electrical* algorithms hardly improved the starting overall score for networks G2 and G3. However, *Dig-Cola* and *IPSEP-Cola* were able to considerably improve the overall score for network G1. We attributed this difference to the much larger number of cycles in G2 and G3 in comparison to G1 (Table 4.1). *CerebralWeb*, an automated method that uses biological information,
achieved scores that were clearly lower than the scores achieved by the hybrid, *Crowd*, *Simulated Annealing*, *Dig-Cola*, *IPSEP-Cola*, and *Spring Electrical* approaches (Figure 4.11). To further understand these results, we examined the downward-pointing paths results in detail.

![Figure 4.12: Distributions of the number of downward pointing paths in the layouts created by different approaches.](image)

**Downward-pointing paths criterion.** Here, we compared the performance of all the approaches on the number of downward pointing pathways, our sole domain-specific layout criterion (Figure 4.12). In Figure 4.12, we saw a trend similar to overall scores shown in Figure 4.11. We attribute this similarity to the high priority we assigned to the DP criterion. This explains that DP is the main reason why our baseline methods did not perform well on networks G2 and G3. We believe as the number of cycles increase in a network, it becomes harder for the automated methods to optimize for downward pointing paths. In fact, *Dig-Cola*, *Spring Electrical*, and *IPSEP-Cola* computed very few downward pointing paths (≤ 10) in networks G2 and G3. In case of CerebralWeb, we noticed that it arranged the nodes in five layers in the following order: “cell surface”, “plasma membrane”, “cytoplasm”, “nucleus”, and “unknown”. While this order has a high-level correspondence with a receptor-to-TF path, CerebralWeb did not lay out the nodes inside each layer in such a way that the final layout
would contain many downward pointing paths. In contrast, crowd workers seem to be able to observe the direction of the flow along the edges and develop moves that significantly increase the number of downward pointing paths despite the presence of cycles. Overall, these results indicate that crowd workers are better than automated methods at downward pointing paths task, specially for networks with several cycles. Moreover, Crowd-SA100 hybrid approach achieves even more number of downward pointing paths than crowd workers alone (Crowd). We discuss the performance of the hybrid approaches in detail later in the section.

Other aesthetic criteria. Next, we compared the distributions of per-criterion scores for the remaining layout criteria of all the approaches (Figure A.7 in Appendix). In these figures, we observed that the simulated annealing (SA) baseline outperformed the Crowd, Hybrid, and other automated approaches on the EC, EL, ND, and NED criteria. However, when we analyzed the median scores achieved by the Crowd and SA (refer to Figure 4.13A), we found that the (positive) percentage difference for the DP criterion was one order of magnitude higher than the (negative) percentage difference for each of the remaining criteria. Since, DP is our sole biology inspired criterion, it is important to achieve high DP score to get a biologically meaning drawing. Therefore, we believe there is an advantage (more biologically meaningful drawing) of using Crowd approach even though it performs slightly worse (less aesthetic) than Simulated Annealing on the EC, EL, ND, and NED criteria. We also observed that the Crowd approach outperformed the Spring Electrical and Dig-Cola methods for all criteria (Figure 4.13B–C). Crowd also outperformed IPSEP-Cola on the DP, ND, and NED criteria (refer to Figure 4.13D).
Crowd versus Hybrid. The Crowd approach outperformed the Crowd-SA100 hybrid approach (Figure 4.13H) on distance-based criteria (EL, ND, and NED). On the other hand, Crowd-SA100 performed better on the DP and EC criteria. We attribute this behavior to assistance from the SA100 simulated annealing component in the Crowd-SA100 hybrid approach. We believe that since SA100 starts at a high temperature, it has the ability to make non-local node movements that could result in the re-orientation of an edge to remove a crossing or to make it downward pointing. These non-local moves allow the Crowd-SA100 approach to use SA100 to further optimize for the DP and EC criteria. In contrast, the hybrid Crowd-SA20 approach performed better than the Crowd approach for distance-based criteria such as EL and ND. Here, SA20 can move a node only to a nearby position and, therefore, allows Crowd-SA20 to explore the local neighborhood of a layout state in a more exhaustive manner compared to crowd workers. Surprisingly, the Crowd approach outperformed the
hybrid Crowd-SA20 approach on the NED criterion, despite assistance in local moves from SA20 (Figure 4.13F). Overall, the results show that the hybrid Crowd-SA100 approach, where simulated annealing makes non-local jumps, performs as well as or better than the non-hybrid Crowd approach.

Figure 4.14: Bar plot of the average improvement in the total score per minute achieved by a crowd worker or a simulated annealing run in the SA, Crowd, and Hybrid approaches.

**Rate of improvement.** Next, to evaluate the rate of improvement, we computed the average improvement in total score per minute achieved by a crowd worker or a simulated annealing run in the SA, Crowd, and Hybrid approaches (Figure 4.14). Here, we observed that the Crowd and Hybrid approaches improved the scores at a faster rate than SA in all three networks, while also considerably outscoring it in networks G2 and G3. We also noted that Crowd-SA100 had a better average improvement in total score per minute than Crowd for all three networks. These results show that while SA and Crowd generate layouts comparable to Crowd-SA100 in some networks, Crowd-SA100 offers a better rate of improvement than these methods. Separately, we also observed that the automated methods — Dig-Cola, IPSEP-Cola, and Spring Electrical— generated the final layout within seconds. The Crowd and Hybrid approaches were slower, but clearly outperformed the automated methods within
4.4. Results

a few minutes. This observation is supported by Figure A.8 in Appendix showing the scores achieved by each approach over time for each network for all task sequences.

Finally, we present the highest scoring layouts generated by Flud and the automated baseline methods as Figures A.1–A.6 and qualitatively compare them in Section A.5 in Appendix.

4.4.3 Q3: Can users with biological training better understand Flud layouts compared to those created by automated methods?

Figure 4.15: Performance results from task effectiveness evaluation described in Experiment 3 (Section 4.3.5). (A) Accuracy (B) Average time taken (in seconds) by the participants to a complete task. (C) Average confidence rating given by the participants on a five-point Likert scale where 5 represents very confident and 1 represents not confident in their answer to the task. (D) Average easiness rating given by the participants on a five-point Likert scale where 5 represents very easy and 1 represents very difficult to perform the task.

Here, we analyzed data collected in Experiment 3 to compare the effectiveness of the highest scoring layouts generated using Flud and the automated baseline methods. To evaluate effectiveness, we measured multiple aspects of participant responses: accuracy, time taken, user’s confidence ratings, and easiness ratings. We tested the normality for all the measures and found that the time taken to finish the task failed the normality assumption (p<0.05).
Therefore, we used the non-parametric Kruskal–Wallis test to analyze these measures. Next, we conducted Mann–Whitney U-tests (a non-parametric alternative to t-tests) for significant Kruskal–Wallis tests. We used Bonferroni corrections to adjust the significance levels for multiple comparisons ($\alpha = 0.05/10 = 0.005$). We also present effect sizes (r) where, $r=0.1$ is a small effect size, $r=0.3$ is medium, and $r \geq 0.5$ is large. We conducted the statistical analyses using R.

**Accuracy.** Figure 4.15A shows the distribution of accuracy achieved by the participants when presented with layouts generated by different methods. Flud (mean=0.92, std=0.1) and Simulated Annealing (mean=0.8, std=0.13) layout condition had high accuracy rates. The other three layout conditions, IPSEP-Cola (mean=0.62, std=0.15), Dig-Cola (mean=0.41, std=0.23), and Spring Electrical (mean=0.4, std=0.14), had medium accuracy rates. Furthermore, the Kruskal–Wallis test showed a significant effect of layout method on task accuracy ($\chi^2(4)=75.534, p<0.001$). Post-hoc Mann–Whitney U-tests showed significant differences in accuracy between Flud and Spring Electrical ($p<0.001, r=0.84$), Flud and Dig-Cola ($p<0.001, r=0.82$), Flud and IPSEP-Cola ($p<0.005, r=0.78$), with large effect sizes, but not between Flud and Simulated Annealing. Overall, the results indicate that Flud creates layouts that lead to more correct interpretations than other baseline methods.

**Time taken.** Next, we compared the time taken by participants to complete the tasks. Figure 4.15B displays the distributions of the average time taken by the participants to complete the tasks under different layout conditions. The Flud condition had the shortest average time in seconds (mean=32.58, std=5.59) compared to Simulated Annealing (mean=43.83, std=5.84), IPSEP-Cola (mean=47.10, std=4.92), Dig-Cola (mean=50.36, std=7.29), and Spring Electrical (mean=52.82, std=2.40). A Kruskal–Wallis test showed a significant effect of layout method on time taken ($\chi^2(4)=96.467, p<0.001$). Post-hoc Mann–Whitney U-tests showed significant differences in time spent on the completed tasks between Flud and Spring
4.4. Results

Electrical ($p<0.001$, $r=0.87$), Flud and *Dig-Cola* ($p<0.001$, $r=0.55$), Flud and *IPSEP-Cola* ($p<0.005$, $r=0.72$), and Flud and *Simulated Annealing* ($p<0.001$, $r=0.60$), with medium and large effect sizes. During the user interviews, the participants indicated that it was harder to identify the order of nodes if the paths were not downward pointing causing them to take more time to complete the task. For example, participant one (P1) said:

> Sometimes the strategy worked, going from the source to the target. Most of the intermediate nodes seem to lie in kind of a straight line connecting the source to the target. If the intermediate nodes were spread out very widely, finding the path tends to be non-trivial. It was too complex to be completed in one minute. ...I needed more than one minute to identify the structure of the path just because the edges are going all over the place...

We also recorded the number of times the tasks timed out, i.e., the participants did not complete the task in 60 seconds. We observed that across all participants, tasks in the Flud layout condition timed out only twice (2.3%) out of 84 recorded submissions. On the other hand, participants timed out for 41 (48.8%) *Spring Electrical* tasks, 25 (29.8%) *Dig-Cola* tasks, 20 (23.8%) *IPSEP-Cola* tasks, and 13 (15.5%) *Simulated Annealing* tasks. Overall, the results indicate that Flud layouts support faster human recognition of paths in networks.

**Confidence.** We also analyzed how confident the participants were with their answer under different layout conditions. Figure 4.15C displays the distributions of average confidence ratings given by the participants. The Flud layout condition received very high confident ratings (mean=4.6, std=0.52) on a five-point scale where "5" corresponds to "very confident" and "1" is not confident. On the other hand, participants were only moderately confident for the *Dig-Cola* (mean=3.10, std=0.76) and *Spring Electrical* (mean=2.82, std=0.68) layout condition. Overall, we observed that the Flud layout condition had the highest confidence...
ratings, compared to *Simulated Annealing* (mean=3.98, std=0.72) and other layout methods. The Kruskal–Wallis test showed a significant effect of layout method on confidence ratings ($\chi^2(4)=71.831, p<0.001$). Post-hoc Mann–Whitney U-tests showed significant differences in confidence ratings between Flud and *Spring Electrical* ($p<0.001, r=0.78$), Flud and *Dig-Cola* ($p<0.001, r=0.87$), Flud and *IPSEP-Cola* ($p<0.005, r=0.59$), and Flud and *Simulated Annealing* ($p<0.005, r=0.32$), with effect sizes ranging from small to large. Interview comments indicated that most participants gave their answers from Flud layouts a rating of 5 (very confident) because the visualizations were easier to understand and clearly portrayed the information flow. For example, participant P3 explained:

I was confident when it was not ambiguous ...if it was ordered in a certain way with the start at the top and the end at the bottom or vice versa. In a lot of problems, the nodes themselves were aligned in like a straight line or roughly a straight line, then the path would also just go straight from the start to the end.

On the other hand, participants gave a lower rating of 3 or 2 if they were not sure about certain edges in the path or not able to complete the task within the given time limit. For example, participant P5 said:

I think when I knew I was not feeling strong about the solution or I just ran out of time. I knew I was wrong because I either wasn’t able to complete it or, I was just trying to forcibly incorporate some nodes in the path....There were times when I wasn’t able to see the edge clearly. It was going through multiple nodes to reach a final node or something. And I wasn’t sure if that was actually correct or not. ...

Overall, the results indicate that participants had the most confidence and clarity with Flud layouts.
Easiness. Next, we compared the easiness ratings given by the participants to understand their self-perceptions of how easy or difficult the tasks were under different layout conditions. The box plot for average easiness ratings given by the seven participants is shown in Figure 4.15D. Here, we observed that the Flud layout condition received very high easiness ratings (mean=4.23, std=0.62) on a 5-point scale, where 5 is very easy and 1 is very difficult. In comparison, IPSEP-Cola (mean=2.96, std=0.17), Dig-Cola (mean=2.74, std=0.35), and Spring Electrical (mean=2.52, std=0.38) were reported as moderately difficult (rating ≈ 3).

Overall, the Flud layout condition had the highest easiness ratings, compared to Simulated Annealing (mean=3.57, std=0.4) and other layout methods. A Kruskal–Wallis test showed a significant effect of layout method on easiness ratings ($\chi^2(4)=95.505$, $p<0.001$). Post-hoc Mann–Whitney U-tests showed significant difference in easiness ratings between Flud and Spring Electrical ($p<0.001$, $r=0.90$), Flud and Dig-Cola ($p<0.001$, $r=0.92$), Flud and IPSEP-Cola ($p<0.005$, $r=0.79$), all with large effect sizes, as well as Flud and Simulated Annealing ($p<0.001$, $r=0.41$), with a small effect size. Participants reported that they selected the easiness rating based on amount of effort they had put in to figure out the answer. For example, P1 said:

There were some times when I was confident that the layout was correct, but the kind of mental strain to get to the point was more.

Participants also reflected on reasons why some visualizations were more difficult than others. For example, P4 talked about how it was harder to follow a path if the nodes were not arranged in downward pointing paths:

...I feel like if the nodes were sort of spread all over and you had to sort of follow them around in a circle and in multiple ways, then that was harder to make sure that I was following it. But if they were like sequential [downward pointing
paths], then that was easier to follow through.

Participants also described other criteria for poorly rating a visualization on the easiness scale, such as many edge crossings, too short or too long edges, and node-node or node-edge overlap. For example, P2 said:

...So when there are many overlapping edges, when the edge is crossed far across the graph, whenever it took me a long time or I ran out of time, then I answered that it was more difficult to read.

Some participants also mentioned that they did not choose very difficult ratings (\textit{easiness} = 1) for any completed task because it was a more suitable rating for the skipped tasks. Participants described these visualizations as having “too many overlapping lines”, “too much overlap” (either node-node or node-edge), “edges are going all over the place”, and “hard to decipher”. Under these circumstances, the participants felt that either they were “completely lost,” or if they had to complete the task, they would have to guess. Therefore, these participants decided to skip tasks with such layouts. Across all participants, tasks with Flud layouts were never skipped or considered hard to understand. On the other hand, the \textit{Dig-Cola} layout condition had the highest skip rate (26.2%), compared to \textit{IPSEP-Cola} (9.5%), \textit{Spring Electrical} (7.1%), and \textit{Simulated Annealing} (2.3%). Overall, these results indicated that participants perceived Flud layouts as easier to understand.

\textit{Need to zoom}. During interviews, participants mentioned the need to zoom in some of the layouts with node-edge overlap. P1 explained:

There were some graphs, particularly where the overlap was the problem, especially the ones where they were long, ...it just seemed that the edges, some of them had nodes close by. We couldn’t really make out arrows unless we zoom
Based on these comments, we conducted a post-hoc analysis of interaction logs to compare the maximum zoom levels used by participants under different layout conditions. The general zoom level in our setup was designed such that initially one pixel in the visualization was equivalent to one pixel on the screen. We observed that the percentage increase in zoom levels was smallest for the Flud layout condition (mean=17.46, std=61.29), compared to Simulated Annealing (mean=53.36, std=212.21), Spring Electrical (mean=132.20, std=1031.53), Dig-Cola (mean=297.86, std=1047.68), and IPSEP-Cola (mean=392.43, std=3438.41). A Kruskal–Wallis test showed a significant effect of layout method on maximum zoom levels used by the participants ($\chi^2(4)=16.365$, $p<0.001$). Post-hoc Mann–Whitney U-tests showed a significant difference in maximum zoom levels between Flud and Dig-Cola ($p<0.001$, $r=0.25$), but not other layout methods.

### 4.4.4 How did the crowd workers optimize the layout scores in the hybrid and crowdsourcing approaches?

Next, we analyzed the data collected from Experiment 2 to understand the nature of the layout process taken by the crowd workers and draw a comparison with Simulated Annealing. We also examined how the crowd workers and Simulated Annealing collaborate with each other and how the hybrid approach impacts the overall performance and their respective contributions.

*Crowds effectively improve overall score.* In order to showcase the crowd workers’ ability to effectively optimize layout scores, we analyzed the number of function evaluations (FEs) [145], a widely accepted meta-heuristic used to compare computational complexity of the optimization approaches. In our case, numbers of FEs is equivalent to number of
Chapter 4. FLUD: a hybrid crowd-algorithm approach

Figure 4.16: Layouts of network G2 at six different stages in an example task sequence, illustrating how crowd workers improve the layout. For illustration purposes, the node and edge widths in these layouts have been modified to highlight representative paths from source node to target node. OS and NumDPP indicates the overall score and the number of downward pointing paths in the given layout.

calls to the layout score function. Our results show that the number of FEs per task sequence for crowd workers in the Crowd (mean=7,436) approach was considerably lower than Simulated Annealing (mean=1,423,080). We also found that crowd workers improved the score in 2.94% of the total FEs. In comparison, Simulated Annealing improved the score in 0.26% of the total FEs. More importantly, we found that the number of FEs required to
improve the layout score by at least 100 points was significantly lower for crowd workers (mean=5, std=21) in comparison to Simulated Annealing (mean=599, std=2716). Overall, the results indicate that crowd workers were more likely to improve the score and found shorter sequences of moves.

*Non-greedy approach.* To illustrate the general approach taken by crowd workers, we present a real example from Experiment 2 which shows the progression of a layout as they improve its score during one of the task sequences (Figure 4.16). We have highlighted some paths from sources to targets in the figure for illustration purposes. In this example, edges between labeled nodes form a cycle. Figure 4.16a shows the layout at the beginning before crowd workers start optimizing it. There are two upward pointing edges in the cycle that need correction. However, these edges are part of a cycle. Therefore, at least one of the edges in the cycle must be upward pointing in every layout. During the crowd task, the worker corrected the red edges in Figure 4.16a and flipped the direction of a downward pointing edge (Figure 4.16d–e) over a sequence of five moves to create more downward pointing paths.

We also noticed that the crowd worker took a non-greedy approach, i.e., the worker initially lost some points (Figure 4.16a–d) before exceeding the initial overall score (Figure 4.16e–f). For ease of discussion, we will now refer to such a sequence of moves as non-greedy, i.e., a sequence of moves where the worker lost some points at least once, and it required at least three moves for the worker to beat the previous high score.

We analyzed such non-greedy moves in the Crowd approach using the data from Experiment 2 to understand their frequency and the impact on the layout scores achieved by crowd workers. Our results show that on average, non-greedy moves accounted for 20% of the instances (mean=45, std=12) where crowd workers improved layout scores in a task sequence. We also found that the length of non-greedy moves for crowd workers (mean=13) was significantly shorter ($t(462)=1.69$, $p<0.001$) than Simulated Annealing (mean=431). The results also
show that improvement in overall scores was significantly higher ($t(405)=2.96$, $p<0.05$) for non-greedy moves (mean=9,779) than otherwise (mean=4,588). In comparison, non-greedy moves in *Simulated Annealing* (mean=985) led to significantly smaller improvement in overall score ($t(405)=2.96$, $p<0.005$). Overall, the results indicate that such non-greedy moves played an integral role in crowd workers’ optimization approach.

*Mixed-initiative collaboration.* To understand the role played by mixed-initiative collaboration, we compared the average improvement to the total score contributed by the crowd workers in different approaches (Figure 4.17), using the data collected in Experiment 2. We found that the contribution by crowd workers is lower in hybrid approaches than in the *Crowd* approach, where the crowd workers work alone without any help from simulated annealing. We also noticed that on average, the contribution by simulated annealing is lower in hybrid approaches than in the *SA* approach for networks G1 and G2. We believe that this decrease in contribution from crowd and simulated annealing in hybrid approaches is because of the distribution of the total work between these two components. However, the *Crowd* and *SA* collaborate with each other in the *Crowd-SA100* approach, leading to higher scoring layouts than either from *Crowd* or from *SA* alone (Figure 4.11).

![Figure 4.17: Bar plot showing average improvement in total score made by simulated annealing and crowd workers in hybrid approach.](image)
We also found that simulated annealing’s contributions in hybrid approaches increased in comparison to SA for graph G3. This observation is supported by Figure 4.17, which shows that the simulated annealing in Crowd-SA100 achieved 1,875% more average improvement in total score than in SA. To highlight the increase in simulated annealing’s performance in Crowd-SA100 in comparison to Crowd, we present an example which shows the progression of the layout as crowd workers and SA100 collaborate back to back during one of the task sequences (Figure 4.18). In this figure, we have highlighted two paths from source to target, for illustration purposes. Figure 4.18a shows the layout at the beginning before crowd workers and simulated annealing start optimizing it. There are many upward pointing edges that need corrections to make the highlighted paths downward pointing. During the crowd task, the worker corrected all of these edges except one (Figure 4.18b). However, one of the highlighted paths was still left with an upward pointing edge. Next, SA100 took over and fixed the other path via two major movements in the layout (Figure 4.18c–d). We argue that SA100 was able to optimize DP in this scenario (Figure 4.18b) because the number of moves required to create a downward pointing path was small in comparison to the starting layout (Figure 4.18a).

We attribute such hybrid collaborations as the main reason behind simulated annealing’s ability to escape local optima (see the nearly vertical lines ending in squares in Figure 4.19) in the hybrid approaches, unlike in the case of SA. Overall, these results indicate the importance of contributions from the crowd workers in the hybrid approach, since SA100 is similar to the early annealing phase in SA (Figure 4.8).

Next, in order to quantitatively characterize the contributions of crowd workers and Simulated Annealing in the hybrid approach, we looked at their average contribution to each per-criterion score in the best performing hybrid approach i.e., CROWD-SA100. We found that crowd’s contribution to the DP score was on average 376% higher (mean=2,421) than
Figure 4.18: G3 network layouts at four different stages in an example task sequence, illustrating how crowd workers and simulated annealing build on each other’s progress with the Crowd-SA100 approach. a) Layout before crowd worker started the task. b) Layout after the crowd worker finished the task. c) Layout during the SA100 annealing process. d) Layout after the SA100 annealing process ends. For illustration purposes, the node and edge widths in these layouts have been modified to highlight two representative paths from source node to target node.
Figure 4.19: Scores achieved by each approach over time for network G3 for a representative sequence. The $x$-axis represents the time taken in hours to reach a particular score and the $y$-axis represents the score. The circle and square markers correspond to crowd workers and simulated annealing respectively.

Simulated Annealing (mean=643). Similarly, we observed that crowds made 192% and 575% higher contribution than Simulated Annealing for EC and NED, respectively. We also found that crowds lost points for EL and ND. Simulated Annealing, on the other hand, gained 1,636 and 222 points on average for EL and ND respectively. These results indicate that crowds are better than Simulated Annealing in optimizing the DP, EC, and NED criteria. In contrast, Simulated Annealing is better at optimizing criteria based on the distance between nodes, i.e., EL and ND.

4.4.5 How did the criterion-specific clue impact use worker’s performance?

We also analyzed the crowd workers’ interaction with the criterion-specific clue and its impact on their performance using the data from Experiment 2. Our results show that when
the crowd workers moved a node not involved in a clue, they lost points and decreased the criterion-specific score (mean=-2.8). In contrast, when crowd workers changed the position of a node that was highlighted in a clue, the criterion-specific score increased (mean=2.9), on average. These results indicate that the criterion-specific clues helped the crowd workers to improve the criterion-specific scores. These observations are supported by Figure A.9 in Appendix.

4.5 Discussion

In this chapter, we presented Flud, a crowd-powered system that allows humans with no expertise to design biologically meaningful graph layouts with the help of algorithmically generated suggestions. Below, we discuss some key takeaways from our evaluation, focusing on overall performance differences, reflections, and design implications.

4.5.1 Flud Facilitates Easy to Comprehend Network Visualizations

Our results showed that network visualizations generated by Flud were easier to comprehend than the visualizations created using state-of-the-art algorithms. Specifically, we found that Flud-generated layouts supported significantly more accurate, faster, more confident, and easier interpretations of signaling flow in biological networks compared to other methods. The results also indicated that one of the main reasons behind the effectiveness of these visualizations is Flud’s ability to lay out more downward pointing paths than the baseline algorithms. The participants in our study reported that it was easier to follow the signaling flow in a visualization if the nodes in a path were laid out sequentially in a top-down manner.
In contrast, if the paths were not downward pointing, participants felt it was non-trivial and more strenuous to follow the signaling flow in the visualization. Algorithmic approaches such as Dig-Cola and IPSEP-Cola focused on optimizing the number of downward pointing edges instead of paths. We found that participants were inaccurate and slow in inferring the information flow from the layouts drawn by these methods. Based on these findings, we argue that merely counting the number of downward pointing edges [154] is not sufficient to capture the flow of information.

Besides lack of downward pointing paths, participants in our study also reported other issues corresponding to the other four layout criteria. Many participants mentioned that it was hard to follow an edge in layouts with too many edge crossings or long edges. Similarly, participants also mentioned that they had to considerably zoom in for analyses, and lacked confidence while inferring edge connectivity in layouts with node-node or node-edge overlap. Baseline algorithmic approaches address some or all of these layout criteria. However, we found that focusing on these criteria alone is not enough to generate useful layouts. Instead, our findings demonstrate that it is important to focus on the proposed downward pointing paths criterion along with the above-mentioned standard criteria to produce effective layouts.

4.5.2 Flud Outperforms Algorithms in Network Visualization Tasks

In our work, we found that the hybrid and crowdsourcing approaches facilitated by Flud were better than state-of-the-art algorithms at five design tasks that are critical for generating high-quality layouts for biological networks. Specifically, our results indicated that crowd workers, in general, are considerably better than the state-of-the-art algorithmic approaches at the downward pointing path task, i.e., laying out downward pointing paths from source nodes to target nodes in a directed network. We believe that there are two main reasons for
the superior performance of crowds.

First, humans are adept at spatial reasoning [219]. These skills allow crowd workers to quickly perceive spatial relations between nodes in the network and detect patterns or scenarios that can be fixed. Second, strategic thinking [219] ability allows humans to plan their actions efficiently. These qualities can allow users to efficiently explore a search space and outperform algorithms at tasks that require visual perception. For example, biological networks have feedback loops, and the crowd workers are able to carefully position the nodes in a feedback loop with respect to each other in order to optimize the number of downward paths. In contrast, an optimization method such as simulated annealing may get stuck in local optima in the presence of feedback loops or cycles, as seen in the network G3.

We also found that algorithmically generated suggestions (clues) played an essential role in the workers’ performance. These suggestions streamlined the broader layout design goal by asking the workers to carefully solve a smaller puzzle (or micro-task), e.g., to move the suggested elements in a certain way in the given network so as to increase the total score. In other words, we were able to reduce the overall challenge level by algorithmically breaking down the layout design task into smaller puzzles that benefit from human judgment and are solvable by untrained crowd workers.

_Hybrid approach outperforms Crowd._ While the _Crowd_ approach was able to outperform the automated methods, the hybrid _Crowd-SA100_ approach offered faster rate of improvement of the score. Moreover, our results indicated that there is a symbiotic relationship between crowd workers and simulated annealing in the hybrid approaches. On the one hand, we believe that the crowd workers improve the layouts such that it is in a state where simulated annealing can make a non-trivial modification to the layout that increases the score. On the other hand, simulated annealing relieves crowd workers of redundant and trivial tasks by improving the layout whenever possible between crowd worker sessions.
4.5. Discussion

4.5.3 Reflecting on Design Decisions

We now discuss some of the decisions we made while planning the scoring system, experiments, and incentives.

**Scoring system**

One key decision we made was to normalize the per-criterion scores used to compare the quality of different layouts. While the downward-pointing paths criterion is new, quantitative metrics for rest of layout criteria have been defined in prior work [154, 155]. However, many of these metrics are on different scales, which makes it challenging to compare layouts of different network sizes. Additionally, we cannot use the same priorities (weights) to compute the weighted layout score for different network sizes. Therefore, we normalized each per-criterion score to lie between 0 and 1, with higher scores being more desirable. This strategy allowed us to use the same priorities for networks G1–G3 in our experiments.

Minimizing the number of edge crossings is one of the most popular aesthetic criteria. It has been empirically validated as an important measure for good network drawing [155, 196, 199].

As an alternative to \( EC \) score, we considered *crosslessness* [128, 154]. We discarded it as an option since its values were very close to \( EC(G) \) for all the networks we used in our experiments. Moreover, we did not see any benefit of using crosslessness over \( EC(G) \) to give feedback to the crowd workers. Hence, we decided to use the \( EC(G) \) metric.

Another design decision we made was to choose a very high priority (400) for the downward pointing paths criterion score (DP) in our experiments. We decided to assign the highest priority to DP because it is our sole domain-oriented criterion, and we wanted it to be preferred over the aesthetic layout criteria in case of conflicts. Moreover, due to our poor approximation of the maximum possible number of downward pointing paths in networks
Chapter 4. Flud: a hybrid crowd-algorithm approach

with cycles, the DP scores were prone to be very low for such networks. If we had selected uniform priorities, due to these low DP scores, algorithms such as Simulated Annealing would choose to optimize for aesthetic layout criteria in case of a conflict, despite even trying the priority of four (greater than the priority of rest of the criteria) for the DP criterion. Therefore, we decided to select a very high priority of 400 so that the weighted contribution of downward pointing paths to the overall layout score is generally higher than other per-criterion scores.

Worker incentives

In this work, we empirically showed that paid crowd workers recruited from Amazon Mechanical Turk can help researchers by successfully laying out networks and achieving high scores that correspond to improved network layouts. We incentivized these workers to submit high-quality layouts by paying them based on their scores [94, 100, 129]. Paid workers offered useful advantages for this research, allowing us to temporarily sidestep the need to build an online community with a critical mass of volunteers while rapidly prototyping and iterating on Flud’s interface in a scaleable, controlled environment. However, prior work in crowdsourcing suggests [25, 162] that unpaid workers or volunteers submit higher quality work in comparison to paid workers. Moreover, the volunteers may be more motivated to perform the task if it supports a worthy cause (for example, scientific research) [110]. Adapting Flud to motivate volunteer participation and present it as a serious game is a promising future direction for this work.
4.5. Discussion

Layout criteria

In this work, we selected five layout criteria based on the direction of flow of information in the network, edge crossings, and relative distances between nodes and edges. We acknowledge that the layouts could be further improved by adding more criteria. However, one of the main goals of the chapter was to empirically show that assigning the per-criterion task modes in the order of their priorities generates better results than assigning them randomly and compare our approach to algorithmic methods. Therefore, we focused on four criteria that are most often used in algorithmic approaches to draw aesthetic graph layouts and a new biologically-motivated criterion to meet the domain-specific needs.

In future work, we plan to extend more layout considerations in Flud such as duplicating nodes in an effort to decrease the number of upward pointing edges and edge crossings, bundling edges that are close to each other and oriented in the same direction, and bending edges [49] to avoid node-edge overlap. In order to facilitate these ideas, we need to extend the types of layout interactions available to crowd workers beyond node manipulation. Designing per-criterion scores, modes, and clues for these layout ideas is a challenging but exciting future direction for our work.

4.5.4 Design Implications

We now discuss some practical implications of our work for constructing systems and methods that support design tasks with numerous conflicting quality constraints.

Mixed-initiative systems are effective in layout design tasks. In our work, we found that humans were able effectively lay out networks. The design used in Flud required humans to have observation and spatial arrangement skills to capture the direction of information flow in the network. Corroborating prior work, we also found that combining the creativity
and intelligence of humans with computational power of automated approaches can lead to high quality results but for a novel layout design task (i.e., 2D network layout that adheres to multiple guidelines). We believe this finding is promising as systems can leverage mixed-initiative settings to tackle other types of design tasks that require spatial arrangement skills. Examples of such tasks include drawing integrated circuits, mapping social networks, and designing interior spaces in houses and buildings.

**Enable preference elicitation to balance multiple criteria.** Balancing multiple criteria in a design task can be challenging for novices. Prior work found that novices are limited by their incomplete knowledge and therefore need preference elicitation support from the system [6]. In the context of this work, crowd workers need to ignore some of the criteria in order to improve the overall layout because they may conflict with each other sometimes. Non-experts can find it difficult to tell which criterion to prefer in case of a conflict. Furthermore, they may find it challenging for them to envision the consequence of preferring one criterion over another. To overcome this challenge, Flud transparently shows the per-criterion priorities, and allows the crowd workers to track changes in the layout score and the corresponding criterion-specific scores with every move. These features provide them with the appropriate preference elicitation required to take resolve conflicts between multiple criteria.

**Computationally assign task modes to break workers out of unproductive task behavior.** Prior work shows that in tasks with multiple sub-goals, specially in human computation games, workers can gravitate towards strategies that are not advised under certain situations [219]. We saw a similar behavior when we asked the workers to focus on the task without fixing a primary criterion specific goal (or mode) at the start of the task. We found that workers gravitate towards distance based criteria such as edge length and node distribution, ignoring the higher priority criterion like downward pointing paths. We
attributed this behavior to workers inexpertise in choosing the appropriate criterion to focus on at a given stage in the task. We overcame this challenge by passing the control to the Flud system and letting it decide a primary criterion for the worker to focus on (known as mode in Flud) during the task. In our work, we empirically showed that assigning the modes in the order of their priorities generates better results than assigning them randomly.

**Automatic suggestions offer guidance when workers are uncertain** Despite having superior intelligence and cognitive skills, workers can sometimes be overwhelmed by the complexity of the task. Under such circumstances, Flud automatically provides workers with a suggestion via the clue feature. These suggestions guide workers by focusing their attention on a smaller, more tractable puzzle inside the overall game. This approach not only reduces the challenge level, but also helps the worker make progress when they get stuck.

### 4.5.5 Limitations

One of the limitations of our evaluation is that the number of networks is small. Since we sought to evaluate multiple task sequences for each of the proposed approaches, we were limited to a small number of networks. A larger number of networks would allow us to show the applicability of the proposed methods to various types of networks. However, the main goals of this chapter were to show that the proposed approaches perform better than the automated methods for networks with large number of cycles and that our results are replicable. Therefore, we used three networks with different numbers of cycles and repeated the game sequence for each approach three times.

Another limitation of our Flud implementation is that its network rendering is currently optimized for small networks (< 100 nodes). Prior work shows that people with no expertise
in network drawing find it difficult to handle large networks [50]. Therefore, we believe that optimizing the system to support larger networks is not the way forward. Instead, we propose to split the larger networks and ask the Flud crowd workers to work on small subnetworks. Yuan et al. [226] proposed a strategy to combine user-generated layouts into a single layout for a large network. Adapting their method to support the downward pointing path criterion for large networks on Flud would be an important future goal.

### 4.6 Chapter Summary

In this chapter, we present Flud, a crowd-powered system for generating high quality visualizations of biological networks. We also introduce a new layout criterion for drawing meaningful biological networks: maximizing number of downward pointing paths. Flud facilitates a novel mixed-initiative technique that combines crowdsourcing with computational methods to achieve high layout scores that correspond to the downward pointing paths criterion and four other aesthetic criteria. We describe how we facilitated the visualization task and made it more accessible to humans, even if they have no biological or computer science expertise. In a crowd experiment, we demonstrate that the benefits of mixed-initiative layout method compared to algorithmic baselines. In another experiment with participants who have biological training, we provide empirical and qualitative evidence of the higher interpretability of Flud-generated layouts compared to algorithmic baselines.

In summary, the contributions of this chapter include:

- A crowd-powered system, Flud, that creates high-quality visualizations of biological networks.
- A novel mixed-initiative approach that combines crowdsourcing with computational
methods to efficiently optimize a layout score/energy function based on the direction of flow of information in the network, edge crossings, and relative distances between nodes and edges.

- A crowd experiment with around 2000 crowd workers shows that: i) assigning the modes in the order of their priorities generates better results than assigning them randomly, and ii) collaboration between crowds and algorithms leads to higher scoring layouts than either from crowds or algorithms alone.

- A mixed-methods study of participants with biological training shows that layouts generated by the mixed-initiative crowd-algorithm approach supported significantly more accurate, faster, more confident, and easier interpretations of information flow in biological networks compared to algorithmic baselines.
Chapter 5

DeepLayout: Laying Out Graphs Using Reinforcement Learning

This chapter is based on a manuscript under review in the journal *Computer Graphics Forum*.


5.1 Motivation

Chapter 4 presents a novel crowd-algorithm approach that combines the creativity and intelligence of the crowd workers to solve complex problems with the computational power of Simulated Annealing algorithm to perform extensive search for global optima. While this approach allows us generate high-quality layouts, the Simulated Annealing algorithm used in this approach has a major limitation: the algorithm does not learn from its past experience; i.e., it does not have the capability to generalize what is a good or bad layout update in the long term for a given graph visualization. Therefore, every time a user executes SA, the algorithm may repeat sub-optimal decisions the same mistakes since it does not utilize the knowledge gained from earlier runs. This same limitation is apparent in all algorithms that aim to create layouts by optimizing an energy-based objective function that encodes the desired aesthetic properties.
5.2 The Graph Layout Optimization Problem

In this chapter, we propose a learning-based approach to graph visualization as an alternative to the Simulated Annealing-based algorithm. Our goal is to train a model to find a graph drawing that optimizes the given layout quality function. Specifically, we want the model to predict a sequence of layout updates for a given graph such that the final layout is optimal for the given objective function. To this end, we propose a reinforcement learning-based technique called DeepLayout. To evaluate DeepLayout’s effectiveness and efficiency in drawing new graphs that are structurally similar to the training data, we conducted an empirical study. The evaluation was conducted for 12 synthetic graph datasets. The graphs in each dataset were structurally similar to each other and generated from directed graphs representing signaling pathways in human cells. We used a layout quality function that similar to the one used in Chapter 4. We compare DeepLayout with state-of-the-art methods like SA [38], Dig-Cola [46], and IPSEP-Cola [48]. Finally, we use qualitative comparisons to demonstrate that the layouts created by our method are superior to those output by a Simulated Annealing-based approach and other state-of-the-art algorithms designed for drawing directed graphs.

5.2 The Graph Layout Optimization Problem

Let $G = (V, E)$ be a directed graph, where $V$ is a set of nodes, and $E$ is a set of edges where each edge is an ordered pair of nodes. A graph is called labeled if every node in the graph has a unique identifier. In this chapter, we consider directed, and labeled graphs. A layout $L_G = \{(x_v, y_v) | v \in V\}$ assigns $x$- and $y$-coordinates to each node in the graph $G$ in order to create a meaningful visual representation.

In this work, we target the graph layout problem considered in energy-based layout approaches [73], in which the goal is to map the nodes of the given graph $G$ onto a bounded
2D layout space, such that final layout optimizes a cost function $C(L_G)$ which encodes the desired layout properties. We refer to this problem as the graph layout optimization problem in the rest of the chapter.

## 5.3 Layout Quality Function

In this chapter, our goal is to create useful graph visualizations for directed networks, especially biological networks. In our prior work [21], we proposed a layout quality function for biological networks that measures the direction of flow of information in the network, the number of edge crossings, and relative distances between nodes and edges. We found that layouts with higher quality scores were easier to understand and supported fast and accurate interpretation of the biological information represented in the graphs. Therefore, we use it as a cost function $C(L_G)$ in the graph layout optimization problem (Section 5.2) to encode the desired layout properties. Specifically, we define the cost as

$$C(L_G) = DP(L_G) + EC(L_G) + EL(L_G) + ND(L_G) + NED(L_G).$$

We now describe the five measures used in the equation.

**Downward Pointing Paths:** This metric quantifies the criterion of maximizing the number of downward pointing paths from source nodes to target nodes in the graph. We compute a normalized downward pointing score as follows:

$$DP(L_G) = \frac{\pi(L_G)}{\rho(L_G)},$$

where $\pi(L_G)$ is the number of downward-pointing paths and $\rho(L_G)$ is the maximum number of downward-pointing paths possible in graph $G$. However, computing $\rho(L_G)$ is a challenging
5.3. Layout Quality Function

combinatorial problem in the presence of cycles. Therefore, we use an approximate upper bound on the number of downward-pointing paths as $\rho(L_G)$. To estimate $\rho(L_G)$, we use a SA algorithm with the number of downward pointing paths as the cost function. The closer $DP(L_G)$ is to one, the larger the number of downward-pointing paths in the layout.

**Non-crossing edge pairs:** Minimizing the edge crossings has been identified as one of the important aesthetic criteria in previous work [21, 87, 154]. Since the maximum number of non-crossing edge pairs possible in $L_G$ is $m(m - 1)/2$, we compute the non-crossing edge pairs score as

$$EC(L_G) = \frac{2\chi(L_G)}{m(m - 1)},$$

where $\chi(L_G)$ is the number of non-crossing edge pairs in layout $L_G$. The closer $EC(L_G)$ is to one, the smaller the number of edge crossings in the layout.

**Edge length:** Edges that are too long or very short can lead to misinterpretations. We consider an edge to be very short if its length $l(e)$ is less than the sum of the radii of the its endpoints (assuming them to be drawn as circles). Therefore, we define the cost $c(e)$ of an edge to be equal to its length $l(e)$ if $l(e)$ is larger than the size of a node. Otherwise, we set $c(e)$ to be equal to a large penalty. Since we normalize the cost of each edge by the largest possible edge length, we use a penalty whose value is larger than the length of the diagonal of the screen. We compute the edge length score of a layout as

$$EL(L_G) = \max \left\{ 0, 1 - \frac{1}{m} \sum_{e \in E} \frac{c(e)}{\sqrt{w_e^2 + h_e^2}} \right\},$$

The closer $EL(L_G)$ is to one, the closer the nodes connected by an edge are, on average.

**Node distribution:** Here, we want to maximize the average distance in the layout between a node and its closest unconnected node. Unlike the previous metric, it applies only to pairs
Figure 5.1: Overview of reinforcement learning approach used in DeepLayout. The agent interacts with the layout environment in discrete time steps. At each time step $t$, the agent first observes the environment as $o_t$, then updates the layout by executing the action $a_t$, and finally receives the change in layout score $r_t$ as a reward for its action.

of nodes not connected by an edge. We compute the node distribution score as

$$ND(L_G) = \frac{1}{n} \sum_{u \in V} \min_{v \in V, (u,v) \notin E} \frac{d(u,v)}{\sqrt{w^2 + h^2}},$$

**Node edge separation:** In this criterion, we want to maximize the average distance between a node and the closest edge to it in the layout. Therefore, we define the node edge separation score as

$$NED(L_G) = \frac{1}{n} \sum_{u \in V} \min_{e = \{t,v\} \in E \, u \neq t \neq v} \frac{d(u,e)}{\sqrt{w^2 + h^2}},$$

where $d(v,e)$ denotes the distance between node $v$ and edge $e$.

### 5.4 DeepLayout Approach

In our work, we propose a deep reinforcement learning-based approach, called DeepLayout, to the graph layout optimization problem (Section 5.2). As our method uses reinforcement learning, we first outline how we map it to the graph layout domain in Section 5.4.1. Then,
we formally define how we represent rewards, state observations, and actions in the reinforcement learning approach in Sections 5.4.2, 5.4.3, and 5.4.4, respectively. We describe the deep neural network architecture in Section 5.4.5. Finally, we discuss the training method used to train DeepLayout in Section 5.4.6.

5.4.1 Formulating Graph Layout as Reinforcement Learning

In our reinforcement learning (RL) approach, we train an agent to sequentially interact with a layout environment and produce a two-dimensional graph drawing, as shown in Figure 5.1. The environment in our case consists of a discretized layout canvas with a fixed number of rows and columns and is formulated as a Markov Decision Process (MDP), with the following key elements:

- Agent: the autonomous component in RL that iteratively observes the information in the environment (including the current state), chooses an action, and receives a reward.

- States $S$: the set of possible layouts. If our discrete canvas has $k$ rows and $l$ columns and $G$ has $n$ nodes, then the size of $S$ is $n^{kl}$.

- Actions $A$: the set of possible actions an agent can take. We restrict an agent to the following actions: i) choose a node and ii) move it to an adjacent grid cell in the discrete layout space.

- Reward $R(s, a)$: the reward for taking action $a$ at a given state $s$. If the agent improves the layout quality, we give it a positive reward proportional to the level of improvement. Otherwise, we give it a negative reward if the layout quality declines.
The goal of the agent is to decide how to update the layout in the environment at discrete
time steps (as shown in Figure 5.1). At each time step \( t \), the agent makes an observation \( o_t \).
For instance, the agent may simply record the current layout \( s_t \); we specify the observation
we use in Section 5.4.3. It then executes the action \( a_t \) resulting from its policy \( \pi \) and receives
a reward \( r_t \) as a consequence of its action.

A policy \( (\pi) \) is a mapping from observations to distributions over actions. It decides the
agent’s behavior at a given time. The goal of RL is to learn a policy \( \pi^* \) that maximizes the
expected discounted sum of immediate rewards \( r_t \) observed over a certain time horizon \( T \).
Mathematically, the optimal policy \( \pi^* \) can be expressed as:

\[
\pi^* = \arg\max_{\pi} \mathbb{E} \left[ \sum_{t=0}^{T} \gamma^t r_t | \pi \right] 
\] (5.1)

The expression being optimized for above is called the \textit{expected cumulative reward}. The
parameter \( \gamma \) is the discount factor (\( \gamma \in [0, 1] \)). It determines the importance of future
rewards. If \( \gamma = 0 \), the agent only considers the current reward and does not account for
future rewards. With a higher value for \( \gamma \), the agent tends to be more far-sighted and
considers future rewards as well. In the extreme case where \( \gamma = 1 \), the RL agent considers
each reward equally. This key parameter can be tuned according to the desired behavior.

The agent’s policy \( \pi(a_t|o_t; \theta), \theta \in \Theta \) and the expected cumulative reward \( \mathbb{E} \left[ \sum_{t=0}^{T} \gamma^t r_t | \pi(a_t|o_t; \theta) \right] \)
depend on a set \( \theta \) of parameters belonging to the parameter space \( \Theta \). In deep reinforcement
learning, these parameters are the internal weights in two neural networks called the policy
network and value network (Figure 5.2). The policy network determines the agent’s policy
and the value network predicts the expected cumulative reward. To create training data for
these neural networks, we actively generate sequences of states, actions, and rewards for a
fixed number \( T \) of time steps (also known as episodes) as follows. Specifically, we start (at
$t = 0$) with a layout generated randomly or by an existing layout method. Then, at every timepoint $0 \leq t \leq T$, we retrieve the observation $o_t$ and use the agent’s current policy to predict an action $a_t \sim \pi(a|o_t; \theta)$. Next, we execute the action $a_t$ to obtain the updated layout $s_{t+1}$ and the reward $r_t$ in return. An episode is a sequence of these $T$ steps. We execute 50 episodes to generate training data which consists of the tuples $(o_t, a_t, \pi(a_t|o_t; \theta), r_t)$ for all timepoints $t$ in the 50 episodes. At the end of the 50 episodes, we post-process the training data to compute the cumulative reward ($\sum \gamma^t r_t$) achieved by the agent after each timestep in each episode. Finally, after this post-processing step, we use the Proximal Policy Optimization (PPO) [175] algorithm to update the model parameters ($\theta$) in the policy and value networks based on the training data. We repeat this process of generating 50 new episodes and updating the model parameters (also known as an epoch) multiple times.

At the start of training, the agent’s policy may predict actions that lead to either negative or positive cumulative rewards. If the agent observes a positive cumulative reward, it perceives the recorded action as a good option for the given observation. Otherwise, it considers it a bad action for the given observation. Over time, the agent learns to take actions that will maximize the cumulative reward.

### 5.4.2 Rewards

The layout quality function, defined in Section 5.3, enables a quantitative comparison of different layouts. Therefore, we use this function to compute the immediate rewards $r_t$. Specifically, the immediate reward $r_t$ at time step $t$ is defined as the change in layout score from time step $t - 1$ to $t$. 
5.4.3 Observations

In DeepLayout, the 2D position of each node in the layout represents the state $s_t$ of the environment at timestep $t$. At each timestep the agent perceives the environment as the observation $o_t$ and makes a decision on how to improve the layout score. In many RL problems, the state and observation are identical. However, in our case, the state of the environment, i.e., the raw node positions, does not contain enough information to support learning a good policy. Many layout quality measures, including the heuristics used in this chapter, do not use raw node positions directly [154, 155]. Instead, these measures are dependent on features such node labels, edge connectivity, and spatial proximity between the nodes. The raw positions alone do not explicitly encode these features.

Furthermore, given the iterative nature of DeepLayout, capturing the current state of the environment alone is not enough to learn an optimal policy. Prior work in RL models for video games [141] has demonstrated the usefulness of the history of observations (typically the last four video frames) seen by the agent. With these considerations in mind, we define an observation such that it captures three types of features for the most recent four layouts seen by the agent. These features include i) node labels, ii) edge connectivity, and iii) spatial relationship between all pairs of nodes. Formally, we define the observation $o_t$ at time $t$ as the concatenation of the feature vectors for the four most recent layouts:

$$o_t = \langle z_t, z_{t-1}, z_{t-2}, z_{t-3} \rangle,$$  \hspace{1cm} (5.2)

where $z_t$ is a vector which captures the features in the layout at time step $t$. This vector $z_t$ is itself a concatenated sequence of node-level features for each node in the layout at time $t$. 

5.4. DeepLayout Approach

These features contain three components:

\[ z_t = (c^1_t, d^1_t, f^1_t, c^2_t, d^2_t, f^2_t, \ldots, c^n_t, d^n_t, f^n_t) \] (5.3)

- \( c^i_t \): A feature vector representing the orientation in the layout at time step \( t \) of node \( i \) with respect to nodes connected to it by an edge in the graph. We use the pairwise cosine distance measure to capture this information. We normalize the pairwise distance by its mean to allow comparison of layouts in different scales. Each vector \( c^i_t \) consists of \( n \) values where the \( j^{th} \) value is equal to normalized cosine distance if \((v_i, v_j) \in E\) or zero, otherwise.

- \( d^i_t \): A feature vector representing the orientation of node \( i \) with nodes not connected to it by an edge in the graph at time step \( t \). Similar to \( c^i_t \), we use a normalized pairwise cosine distance measure to capture this information. Each vector \( d^i_t \) consists of \( n \) values where the \( j^{th} \) value is equal to normalized cosine distance if \((v_i, v_j) \notin E\), otherwise zero if \((v_i, v_j) \in E\).

- \( f^i_t \): A scalar value representing the node type (source, target, or intermediary) for node \( i \).

Overall, the number of features \((c^i_t, d^i_t, f^i_t)\) is \( 2n + 1 \) for each node. Since there are \( n \) nodes in the graph, the size of the observation vector is \( 4(2n^2 + n) \). However, the problem with this representation is that the size of the observation vector is dependent on the number of nodes in the graph. In order to support the layout of graphs with different sizes, we fix the size of the observation vector to \( 4(2k^2 + k) \), where \( k \) is the upper limit on the number of nodes in the target graphs we want to draw. In graphs where \( n \) is less than \( k \), we set the extra entries in node-level features to zero.
5.4.4 Actions

At each time step $t$, the RL agent decides on an action $a_t \in A$ using its policy $\pi(a_t|o_t; \theta)$. Prior research shows that training a RL model with a large and continuous action space is a difficult challenge \[44, 107, 198\]. Therefore, to lower the complexity of the learning problem, we convert the canvas into an $m \times m$ grid such that the action taken by the agent is discrete in nature. We further limit the size of the action space by restricting a node’s movement to only adjacent cells to the left of, right of, above, or below the current position. Finally, we restrict the agent to move only one node at a time. In summary, the agent’s action makes two decisions: select one of the $n$ nodes and choose one of the four possible adjacent grid cells as the new position in the discrete layout space. Therefore, for any given state, the agent can take one of $4n$ possible actions or placements of nodes in their neighborhood in $m \times m$ layout grid (as shown in Figure 5.2).

The agent’s policy $\pi(a_t|o_t; \theta)$ is modeled by the policy network in our architecture (Figure 5.2). The output of the policy network is the probability distribution of the $4n$ possible actions. The predicted action $a_t$ is the action with the highest probability.

5.4.5 Architecture

We design a deep neural network architecture and use RL to train it. It consists of two parts: a policy network to generate a probability distribution over actions and a value network to compute the expected cumulative reward (also known as state value) for a given state. Figure 5.2 shows an overview of the architecture of the policy network and value network architecture. The observation layer encodes information from the adjacency matrix, node type, and node positions for the latest 4 time steps. It feeds a classical feedforward deep neural network (DNN) with tanh activation functions. The output of the feedforward
5.4. DeepLayout Approach

Figure 5.2: Policy and value network architecture used in DeepLayout. The observation layer encodes data from adjacency matrix, node type, and node positions for the latest 4 time steps. The policy and value networks then output a probability distribution over possible actions and an estimate of the expected cumulative reward for the current layout, respectively. FC stands for fully connected layer.

network goes into the policy network, which is composed of a fully connected layer (FC) and a softmax layer. The output of the same feedforward network also feeds to the value network, which is composed of a feedforward network with two fully connected layers.

5.4.6 Training Method

As mentioned earlier (Equation 5.1), DeepLayout’s objective is to train a policy network $\pi(\theta)$ that maximizes the expected value of the reward over the policy network’s placement distribution. To optimize the parameters of the deep neural network, we use Proximal Policy Optimization (PPO) [175] with clipped surrogate objective loss $L_{CLIP}^{VF+H}(\theta)$. Before defining this loss function, we describe its three components: the policy loss function (Equation 5.4), the value loss function (Equation 5.5), and the entropy term $H$. We define the
policy loss as follows:

\[ L_{t}^{CLIP}(\theta) = \min(r_{t}(\theta)\hat{A}_{t}, \text{clip}(r_{t}(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_{t}) \]  

(5.4)

Here \( \hat{A}_{t} = V_{t}^{\text{target}} - V(o_{t}; \theta) \) is the estimated advantage at timestep \( t \) of taking action \( a_{t} \) when the agent perceives the environment as the observation \( o_{t} \). We compute it as the difference between the cumulative reward observed in the training data \( (V_{t}^{\text{target}}) \) and the cumulative reward estimated by the value network \( (V(o_{t}; \theta)) \) at timestep \( t \). The value \( r_{t} = \pi(a_{t}|o_{t}; \theta)/\pi(a_{t}|o_{t}; \theta_{\text{old}}) \) is the ratio between the probabilities of taking action \( a_{t} \) for \( o_{t} \), as determined by the new policy and by the old policy; \( r_{t} \) allows us to capture the change between two policies. The first term in the policy loss function \( (r_{t}(\theta)\hat{A}_{t}) \) is the product of these two values. Maximizing this product encourages learning a higher probability for a good action and a lower probability for a bad one [174]. The second term, \( \text{clip}(r_{t}(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_{t} \) is the same as the first one, except that we restrict \( r_{t}(\theta) \) to lie between \( 1 - \epsilon \) and \( 1 + \epsilon \). Using the clipping parameter \( \epsilon \) ensures that we do not learn a new policy that deviates considerably from the current policy too much. We take a minimum of these terms so that final policy loss function \( L_{t}^{CLIP}(\theta) \) is a pessimistic bound on the first term.

The value loss function \( L_{t}^{VF}(\theta) \) defined below computes the squared loss between the cumulative reward observed in the training data \( (V_{t}^{\text{target}}) \) and estimated by the value network \( (V(o_{t}; \theta)) \).

\[ L_{t}^{VF}(\theta) = (V_{t}^{\text{target}} - V(o_{t}; \theta))^{2} \]  

(5.5)

Finally, we combine the policy and value loss functions in \( L^{CLIP+VF+H}(\theta) \) since they share parameters in our neural network architecture. The third component in \( L^{CLIP+VF+H}(\theta) \), \( H \), represents the entropy of the probability distribution over possible actions. We add the entropy term to encourage exploration of different policies [175]. Finally, we combine the
three components $L_{t}^{CLIP}$, $L_{t}^{VF}$, and $H$ to compute the clipped surrogate objective loss as

\[ L^{CLIP+VF+H}(\theta) = \mathbb{E}_{t}[L_{t}^{CLIP}(\theta) - c_{1}L_{t}^{VF}(\theta) + c_{2}H(\pi(o_{t}; \theta))], \]

where $\mathbb{E}_{t}$ represents the expected value at timestep $t$, and $c_{1}$, $c_{2}$ are weights that quantify the relative importance of the value loss function and entropy term, respectively.

5.5 Evaluation

This section describes the evaluation datasets and experiment setup.

5.5.1 Measuring Structural Similarity between Graphs

In order to ensure that the graphs in our evaluation datasets are structurally similar to each other, we compute the similarity between each pair of graphs in a dataset. There exist many approaches to compute similarity between graphs such as edit distance [68, 166] and network statistics based on degree distributions [143]. One such popular approach for structurally comparing graphs is graphlet frequency similarity. This idea was first introduced by Pržulj et al. as a similarity measure that captures the local structural properties of a graph [153]. A graphlet is defined as a small induced subgraph of a large network. Prior research shows that graphlets can be used to define family of graphs [128, 140] in many domains including social networks [95] and biological networks [152, 168, 223]. In this work, we compute a frequency vector $x$ for a given network by computing the relative frequency $x_{i}$ of each 2– to 4–node directed graphlet $i$ defined by Sarajlic et al. [168] (40 in total). Specifically, we first count the occurrence $f_{i}$ for each graphlet $i$ in the given network. Then, we logarithmically scale the frequencies using the following definition: \[ x_{i} = \log\left(\frac{f_{i} + 1}{\sum(f_{i} + 1)}\right). \] Finally, we use
Laplacian kernel \( \langle x, x' \rangle = \exp\left(-\frac{||x-x'||}{\sigma}\right) \) to compute the similarity between two graphs with frequency vectors \( x \) and \( x' \) \([126]\). We set \( \sigma \) as the length of the frequency vector.

### 5.5.2 Generation of Graph Datasets

In order to evaluate DeepLayout’s ability to generate layouts for new graphs that are structurally similar to the training data, we synthetically create 12 datasets. The graphs in each dataset are directed, labelled, and structurally similar to each other. In all graphs, we use the receptor and transcription factor proteins as sources and target nodes, respectively. Moreover, we generate the datasets such that they capture graphs with different number of edges, paths from sources to targets, and cycles (columns 2–5 of Table 5.2).

Specifically, we use a directed graph that represents a signaling pathway in human cells (the IL-2 pathway in the NetPath database \([105]\)) and randomly modify a fixed percentage of edges (1%, 10%, 25%, and 50%) to generate similar graphs. We use three types of modifications: deletion, addition, and reorientation. Deletion (‘del’) removes an edge between a connected pair of nodes and therefore decreases the number of source-to-target paths and cycles in the graph. In contrast, addition (‘add’) creates a new directed edge between a disconnected pair of nodes and boosts the number of source-to-target paths and cycles. Reorientation (‘flip’) flips the direction of an edge between a connected pair of nodes. In contrast to addition and deletion, reorientation can either add or remove source-to-target paths and cycles to the graph.

Overall, we generate 20 graphs in each of the 12 datasets. We note that the structural similarity between graphs in each dataset is greater than 90% (Section 5.5.1). For evaluation, we split each dataset randomly into two equally-sized sets of training and testing graphs. We later use the training graphs to train DeepLayout. Specifically, while training DeepLayout,
5.5. Evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dig-Cola [46]</td>
<td>level gaps = [-2, -1, -0.1, 0.1, 1, 2]</td>
</tr>
<tr>
<td></td>
<td>edge length = [0.1, 1, 3.125, 5]</td>
</tr>
<tr>
<td></td>
<td>ratio = ['fill', 'compress', 'auto']</td>
</tr>
<tr>
<td></td>
<td>overlap = ['scalexy', 'compress']</td>
</tr>
<tr>
<td>IPSEP-Cola [48]</td>
<td>level gaps = [-2, -1, -0.1, 0.1, 1, 2]</td>
</tr>
<tr>
<td></td>
<td>edge length = [0.1, 1, 3.125, 5]</td>
</tr>
<tr>
<td></td>
<td>ratio = ['fill', 'compress', 'auto']</td>
</tr>
<tr>
<td></td>
<td>overlap = ['scalexy', 'compress', 'ipsep']</td>
</tr>
<tr>
<td></td>
<td>sep = [9, 5, 1]</td>
</tr>
</tbody>
</table>

Table 5.1: The layout methods and parameter ranges used for producing layout for graphs in testing datasets. The parameter names follow the documentation of the implementations.

we randomly pick a graph from the training set at the start of each episode. Finally, after the training is over, we use testing graphs to evaluate DeepLayout.

5.5.3 Baseline Methods

To showcase the effectiveness of DeepLayout, we compare it with two well-known layout algorithms that consider edge direction — IPSEP-Cola [48] and Dig-Cola [46]. To make comparisons fair, we run a hyper-parameter sweep for these method on training graphs, select the parameter set that generates the highest scoring layout for a given dataset, and use it to generate layouts for the remaining (testing) graphs. We provide the list of values used in the hyper-parameter sweep for each layout method in Table 5.1.

Additionally, we evaluate SA as a baseline method. Like reinforcement learning, SA can be used to optimize arbitrary non-differentiable objective functions [3, 193]. In our evaluations, we use the same starting layouts and layout quality function as RL to generate final layouts. We run SA for 100 iterations with 10 layout updates each. After each iteration, we decay the temperature by a factor of 0.95. However, unlike reinforcement learning, we do not restrict node movements in SA to neighboring cells.
5.5.4 Model and Training Configurations

In our experiments, we set \( k = 40 \) (Section 5.4.3) in DeepLayout as the number of nodes in training and testing graphs is at most 40. We train the model with Adam optimizer with a learning rate of 0.0001. We use 256 hidden units in first two fully connected layers. We use 0.6 as the discount factor (\( \gamma \)). To compute the loss, we use \( c_1 = 0.5 \) and \( c_2 = 0.01 \) as coefficients of the value loss function and entropy term respectively. We set the size of training data as 50 episodes and the length of an episode \( T \) (Equation 5.1) as 1,000 timesteps.

We create all the visualizations on a layout canvas of size of 5000 × 6000 pixels, which is appropriate for laying out the graphs used in this chapter. There are many choices to discretize the 2D layout canvas into grid cells. This decision impacts the difficulty of the graph layout optimization problem and the quality of the final layouts. We discretize the layout canvas into 20 rows and columns. We draw each node with a size of 250 × 250 pixels, which is big enough to render easy-to-read node labels. We use the layouts generated by Dig-Cola as the starting point for both DeepLayout and SA.

5.5.5 Implementation

DeepLayout is implemented in PyTorch. We used the Dig-Cola and IPSEP-Cola implementation in the neato program in the Graphviz package. The SA baseline method is implemented in Python. We conducted the experiments on a machine with an Intel Xeon Gold 6136 (28 cores at 3.0GHz) CPU and an NVIDIA V100 (Volta) GPU.
5.5. Evaluation

5.5.6 Results

In this section, we evaluate our method and answer the following questions. How does the quality of the generated layouts compare to state-of-the-art baselines? How does the size of the training dataset impact the model performance? How does various hyper-parameter choices impact model performance?

Quantitative Comparison with Baseline Methods

We compare the quality of layouts and running time for DeepLayout to three state-of-the-art baseline methods: SA, Dig-Cola, and IPSEP-Cola. To train DeepLayout, we use 50% of the graphs in each dataset and evaluate their performance on the remaining graphs. Subsequently, we evaluate the performance of the baselines on the test graphs. Overall, the results show that for 9 out of 12 datasets, DeepLayout computes layouts with higher scores than layouts created by the baseline approaches (Table 5.2). We also observe that for the datasets with the smallest number of changes (del–1, add–1, flip–1), DeepLayout displayed the highest percentage difference over the score achieved by Dig-Cola, which we used to generate the initial layout for DeepLayout and SA. For the remaining datasets (del–10, del–25, del–50), we observe that although DeepLayout considerably beats Dig-Cola and IPSEP-Cola, SA achieves better score for these datasets. We attribute this better performance by SA to the simplicity of the graphs in these datasets. Specifically, we note that graphs in these datasets had the fewest edges and cycles, and the lowest estimated upper bound on the number of downward pointing paths (ρ). In contrast, DeepLayout was the best performing method in the nine datasets where the average number of cycles greater than four.

Comparison with Simulated Annealing: Further analysis showed that compared to DeepLayout, SA struggled to generate high-quality layouts, including 34.5% fewer down-


Table 5.2: Comparison of the average best scores achieved by DeepLayout, SA, Dig-Cola, and IPSEP-Cola for 12 synthetically generated graph datasets. The values in parentheses indicate the average percentage difference versus the scores achieved by Dig-Cola (used as the starting layout for DeepLayout and SA). The highest layout scores are boldfaced. $|V|$: average number of nodes, $|E|$: average number of edges, $\rho$: average estimated upper bound on the number of downward pointing paths, cycles: average number of simple cycles.

| Dataset | $|V|$ | $|E|$ | $\rho$ | cycles | DeepLayout | SA | Dig-Cola | IPSEP-Cola |
|--------|------|------|-------|--------|-----------|----|----------|-----------|
| del–1  | 40   | 73   | 95    | 5      | 22,547    | 21,198 | 18,085   | 16,876    |
| del–10 | 38   | 67   | 55    | 4      | 20,217    | 21,500 | 18,835   | 17,708    |
| del–25 | 35   | 56   | 35    | 2      | 20,588    | 22,051 | 18,214   | 18,388    |
| del–50 | 32   | 37   | 5     | 1      | 23,740    | 25,528 | 20,386   | 18,889    |
| add–1  | 40   | 75   | 109   | 6      | 23,776    | 20,951 | 17,968   | 17,506    |
| add–10 | 40   | 82   | 157   | 20     | 20,845    | 20,245 | 18,220   | 15,854    |
| add–25 | 40   | 93   | 270   | 119    | 20,026    | 19,160 | 18,521   | 16,852    |
| add–50 | 40   | 111  | 1,669 | 16,052 | 17,855    | 17,817 | 16,902   | 17,822    |
| flip–1 | 40   | 74   | 104   | 7      | 21,976    | 20,941 | 17,975   | 16,737    |
| flip–10| 40   | 74   | 130   | 25     | 20,557    | 19,136 | 17,501   | 15,837    |
| flip–25| 40   | 74   | 194   | 72     | 19,280    | 19,093 | 18,035   | 16,052    |
| flip–50| 40   | 74   | 86    | 92     | 21,113    | 20,528 | 19,265   | 15,208    |

Comparison with Dig-Cola and IPSEP-Cola: Next, we compare our results with the state-of-the-art methods Dig-Cola and IPSep-Cola. In terms of speed, we note that Dig-Cola and IPSep-Cola are substantially faster than DeepLayout. They generate the final layout in less than a second, compared to 20 seconds on average for DeepLayout. However, DeepLayout holds the following advantages over them with respect to quality. First, it is able to adhere to domain-specific design constraints, such as maximizing the downward pointing paths from source to target nodes. Second, its layout scores improve with training as the model perceives the returns (positive or negative rewards) for taking a sequence of possible actions. Finally, it uses a policy search method (PPO [175]) that allows us to optimize various non-differentiable layout scoring functions [8, 206], without the need to formulate...
### 5.5. Evaluation

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Timing (seconds)</th>
<th>Number of Downward Pointing Paths</th>
<th>Number of Crossings</th>
<th>Average Edge Length (pixels)</th>
<th>Number of Node Overlaps</th>
<th>Number of Node Edge Overlaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>del–1</td>
<td>21</td>
<td>44</td>
<td>48</td>
<td>21</td>
<td>1,091</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>21</td>
<td>48</td>
<td>21</td>
<td>1,091</td>
<td>5</td>
</tr>
<tr>
<td>del–10</td>
<td>12</td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>902</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>12</td>
<td>18</td>
<td>18</td>
<td>902</td>
<td>3</td>
</tr>
<tr>
<td>del–25</td>
<td>13</td>
<td>29</td>
<td>13</td>
<td>17</td>
<td>952</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>13</td>
<td>17</td>
<td>17</td>
<td>952</td>
<td>5</td>
</tr>
<tr>
<td>del–50</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>877</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>877</td>
<td>3</td>
</tr>
<tr>
<td>add–1</td>
<td>23</td>
<td>37</td>
<td>68</td>
<td>37</td>
<td>1,111</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>68</td>
<td>37</td>
<td>37</td>
<td>1,111</td>
<td>4</td>
</tr>
<tr>
<td>add–10</td>
<td>25</td>
<td>35</td>
<td>48</td>
<td>37</td>
<td>1,044</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>48</td>
<td>37</td>
<td>37</td>
<td>1,044</td>
<td>5</td>
</tr>
<tr>
<td>add–25</td>
<td>33</td>
<td>24</td>
<td>68</td>
<td>35</td>
<td>1,161</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>68</td>
<td>35</td>
<td>35</td>
<td>1,161</td>
<td>4</td>
</tr>
<tr>
<td>add–50</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>976</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>976</td>
<td>2</td>
</tr>
<tr>
<td>flip–1</td>
<td>26</td>
<td>42</td>
<td>47</td>
<td>36</td>
<td>1,004</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>47</td>
<td>36</td>
<td>36</td>
<td>1,004</td>
<td>5</td>
</tr>
<tr>
<td>flip–10</td>
<td>21</td>
<td>26</td>
<td>38</td>
<td>15</td>
<td>1,072</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>38</td>
<td>15</td>
<td>15</td>
<td>1,072</td>
<td>7</td>
</tr>
<tr>
<td>flip–25</td>
<td>21</td>
<td>15</td>
<td>29</td>
<td>18</td>
<td>973</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>29</td>
<td>18</td>
<td>18</td>
<td>973</td>
<td>4</td>
</tr>
<tr>
<td>flip–50</td>
<td>24</td>
<td>26</td>
<td>14</td>
<td>13</td>
<td>1,060</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>14</td>
<td>13</td>
<td>13</td>
<td>1,060</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5.3: Comparison of DeepLayout versus Simulated Annealing (SA). The results report the average values observed over 5 runs for each graph in a dataset. The highest values are boldfaced.

differentiable equivalents of those functions. As a result, we notice that DeepLayout is substantially more effective in generating high-quality layouts that meet design criteria (14.3% higher layout scores on average).

#### Qualitative Comparison with Baseline Methods

Figure 5.3 shows the layouts generated for a representative graph in the add–1 dataset. Recall that the layout from Dig-Cola is used as a starting layout for DeepLayout and for SA. We observe that it is relatively easier to interpret source-to-target paths in layouts generated by DeepLayout in comparison to SA and Dig-Cola. For example, it is easy to notice that source node 26 directly influences the target node 36 in all layouts. However, it is very hard to interpret the indirect influence of source node 26 on target node 36 in the SA and Dig-Cola layouts. In contrast, DeepLayout’s layout clearly shows this influence through two edge-disjoint paths: \(26 \rightarrow 17 \rightarrow 0 \rightarrow 36\) and \(26 \rightarrow 23 \rightarrow 38 \rightarrow 32 \rightarrow 36\).

We also observe that there are two and one node overlaps in the Dig-Cola and SA layouts, respectively. In contrast, the layout created by DeepLayout does not have any node overlaps. We also notice that many connected pairs of nodes are very close to each other in the
Figure 5.3: Layouts generated by DeepLayout, SA, and Dig-Cola, respectively for a representative graph in the add–1 dataset. We do not show the layout from IPSEP-Cola, which achieved the lowest score, due to space constraints.

Dig-Cola and SA layouts. Therefore, it is very hard to determine the connectivity and the direction of the edges among these nodes. In contrast, the edges are longer and easier to read in layouts generated by DeepLayout.

5.5.7 Training with Larger Datasets

In order to understand the impact of the size of the training dataset, we synthetically generate 70 more graphs for 3 different datasets (del–1, add–1, flip–1). Then, we train the DeepLayout on these datasets with 5 different training sizes: 10, 20, 40, 60, and 80 training graphs. We ensure that the datasets with fewer training graphs are subsets of larger datasets. Next, we evaluate these models on a different set of 10 graphs in each dataset. Figure 5.4A shows the impact of training DeepLayout with a larger dataset.

We observe that as we train with more graphs, we are able to generate higher quality layouts (left plot in Figure 5.4A) with faster speeds (right plot in Figure 5.4A). We believe that the lower layout scores achieved by the model trained on 10 graphs is due to overfitting. As we
5.5. Evaluation

5.5.8 Impact of Hyper-parameters Choices

The choice of model hyper-parameters (e.g., layout grid size, number of hidden layers) in DeepLayout may impact its performance. Therefore, we conduct experiments to understand their impact.

Figure 5.4: DeepLayout’s performance on three different training datasets (del–1, add–1, flip–1) trained by varying different hyper-parameters. A) number of training graphs, B) number of layout grid cells in the discrete layout, C) number of hidden layers in the deep neural network architecture (Section 5.4.5), D) layout feature vector observed by the RL agent (Section 5.4.3) with the layout cost function defined in Section 5.3, and E) layout feature vector observed by the RL agent with the layout cost function without DP score. The bar plots titled ‘Overall Score’ show the scores achieved by DeepLayout models trained under different settings. The bar plots titled ‘Timing’ show the time taken to reach the highest scoring layout by the models trained under different settings. The error bars represent the 95% confidence interval.

expose the model to more graphs while training, it becomes less prone to overfitting and therefore achieves better layout score and running time.
Layout Grid Size

In DeepLayout, we discretize the 2D layout canvas into grid cells to reduce the complexity of the learning problem. However, this decision impacts the quality of the final layouts as well. Therefore, we evaluate the DeepLayout’s performance on 3 different datasets (del–1, add–1, flip–1) at three different resolutions of the layout grid: $20 \times 20$, $25 \times 25$, and $40 \times 40$. Figure 5.4B shows the impact of these choices on DeepLayout’s performance. Here we evaluate each model on a testing dataset with 10 graphs.

The results show that the model trained with the $25 \times 25$ layout grid achieves slightly higher layout scores (2.8% on average) than the model trained with $20 \times 20$ layout grid for add–1 and flip–1 dataset (left plot in Figure 5.4B). We also observe that the model trained with the $25 \times 25$ layout grid reaches its optimal layout at a faster speed than the model trained with $20 \times 20$ layout grid (right plot in Figure 5.4B). However, when we increase the number of layout grid cells to $40 \times 40$, the improvement in layout scores diminishes (1.6% on average) and layout time increases. This increase in time is due to an increase in the number of layout updates required to reach the optimal layout from the starting layout. Overall, these results indicate that increasing the grid resolution leads to minor improvements in layout score but has substantial effect on running time.

Number of hidden layers

We evaluate DeepLayout’s performance with different numbers of hidden layers (1, 2, and 3). Figure 5.4C shows the impact of number of hidden layers on DeepLayout’s performance. The figure shows that the layout scores decrease as we increase the number of hidden layers. We attribute this decline in performance to overfitting.
5.5. Evaluation

**Layout Feature Vector**

In DeepLayout, we capture the spatial relationship between nodes in the input layout feature vector using the cosine distance measure. In this experiment, we evaluate DeepLayout’s performance on two other input layout feature vectors. First, we construct the input layout feature vector using the Euclidean distance measure. Second, we construct the input layout feature vector by concatenating the feature vectors with cosine distance and Euclidean distance. We train DeepLayout for three different training datasets (del–1, add–1, flip–1) with each of these choices. Figure 5.4D shows the impact of input feature vector on DeepLayout’s performance where each model is evaluated on a testing dataset with 10 graphs. The results show that overall scores achieved by the model trained with the cosine distance-based feature vector are higher than the model trained with the Euclidean distance-based feature vector. We also observe that the overall scores do not improve if concatenate these feature vectors.

5.5.9 Cost Function Design

Next, we train and evaluate DeepLayout with a cost function without the downward pointing paths layout criterion. Figure 5.4E shows DeepLayout’s performance on del–1, add–1, flip–1 datasets where each model is evaluated on a testing dataset with 10 graphs. We note the substantial decrease in scores in comparison to Table 2 (4,776 on average for DeepLayout) is due to absence of downward pointing paths layout criterion. In its absence, we observe minor differences in score in Figure 5.4E but substantial change in running time with different methods.

Specifically, we found that DeepLayout’s performance slightly improves (0.4% on average) if we use Euclidean distance measure instead of cosine distance measure in the input feature vector (Figure 5.4E). Subsequently, we compare the performance of DeepLayout trained us-
ing Euclidean distance based input feature vector with the baseline methods. In this analysis, we observe that the layout scores achieved by DeepLayout are only slightly better (0.41% on average) than SA, the best performing baseline method. However, DeepLayout converges faster than SA and is on average 60.69% faster than SA. In contrast, the DeepLayout model trained with the cosine distance-based input feature vector was only 49.1% faster than SA.

5.6 Discussion

Our evaluation shows that DeepLayout can learn a generalized layout process that optimizes a given cost function to generate drawings for new graphs. In this section we discuss the advantages and limitations of the proposed approach for graph drawing.

A major advantage of DeepLayout is that it uses an energy-based layout approach where all it needs is a cost function that encodes desirable layout properties. The advantage of using a cost function is that it can not only include well-known aesthetic heuristics [38, 154, 155] but also be easily adapted and extended to fulfill additional domain-specific needs (e.g., maximizing downward pointing paths). In contrast, training with layout examples (e.g., DeepDrawing [216]) comes with the additional overhead of finding or generating graph drawings for training. DeepDrawing required as many as 26,000 algorithmically created layout examples to train its model to draw the complex graphs used in their experiments. In a real world scenario, finding such a large number of good layouts will require a lot of human effort. DeepLayout overcomes the need for layout examples by using reinforcement learning to actively generate its training data as part of the training process.

Once DeepLayout is trained, it can be directly used by novices without any parameter tuning. However, our approach needs expert intervention before training the model. Specifically, the layout cost function needs to be carefully designed such that it captures the domain-specific
5.6. Discussion

needs and aesthetic requirements. This step requires expertise in graph drawing and domain knowledge represented in the networks to be laid out. Additionally, we also need experts to specify the hyper-parameters while training the model. In this work, we provide a summary of the lessons we learned while evaluating the impact of various hyper-parameters in DeepLayout. We hope it will assist experts to train DeepLayout models for their networks.

Our results also show that DeepLayout can be trained for two different cost functions: one considers a criterion based on path orientations and the other ignores it. In contrast, other machine learning-based methods (e.g., DeepDrawing) ignore edge orientation. Most importantly, we show that DeepLayout is good at optimizing the downward pointing paths criterion in networks with many cycles, which is hard to achieve with any other type of algorithm.

Although our model is capable of generalizing the layout process for similar types of graphs for a given layout cost function, it does not generalize to graphs that are dissimilar to the training data or to a different cost function. Training one model for different type of graphs so that we can use it for any unseen graphs is a promising direction for future research. Another limitation of our approach is that it is not deterministic. Since DeepLayout samples the actions from the probability distribution over possible actions predicted by the policy network, it may not produce the same layout in every run. On the other hand, an advantage is that the user can run DeepLayout many times on a new network and get a palette of layouts to choose from.

In this work, we show that DeepLayout is in general better than Simulated Annealing in networks with many (> 4) cycles. In comparison to Flud which optimizes a similar layout scoring function, DeepLayout is fully automated and can quickly lay out the network. That said, we note DeepLayout’s performance is limited by the sequences of layout actions seen during the training process. As a result, even though DeepLayout created more downward pointing
paths than SA in datasets with large numbers of cycles (add–25, add–50, flip–25, flip–50), we noticed that there is still scope for further improvement in the score. An implication of this finding is that we can use DeepLayout in place of SA in Flud’s mixed-initiative approach to generate layouts with more downward pointing paths which are important for creating effective layouts. Integrating DeepLayout into Flud and characterizing how DeepLayout and crowds may complement each other are promising future directions of this work. Another direction that warrants further exploration is using DeepLayout to learn how crowds create layouts and later reuse this information to generate visualizations for structurally similar graphs.

Finally, as shown in Table 5.2, we evaluated DeepLayout on small graphs (< 40 nodes). To train DeepLayout for larger graphs, one must adjust the number of grid cells and output size of the final layer in the policy network. However, training a model for larger graphs poses multiple challenges. First, the action space increases with number of nodes. Learning a policy for larger action spaces is harder and takes more training time. Second, increasing the number of nodes or edges can significantly increase the training time, since we need to iteratively compute rewards during the training process. Adapting the method to support large networks is an important future goal.

5.7 Chapter Summary

In this chapter, we present DeepLayout, a graph layout approach with the ability to generalize, i.e., it has the ability to learn from a layout process to generate drawings for new, unseen graphs. In an empirical study, we demonstrate that DeepLayout outperforms state-of-the-art methods like Dig-Cola, IPSEP-Cola, and Simulated Annealing. We also show that DeepLayout was, on average, 23.6% faster than the Simulated Annealing method.
5.7. Chapter Summary

In summary, the contributions of this chapter include:

- A graph layout approach, DeepLayout, that uses reinforcement learning to learn the layout strategy for efficiently optimize a layout score/energy function based on the direction of the flow of information in the network, edge crossings, and relative distances between nodes and edges.

- An evaluation of 12 synthetic graph datasets generated from a biological network showing that DeepLayout outperforms the Simulated Annealing algorithm used in Flud’s mixed-initiative approach in 9 out of 12 datasets.

- An evaluation of the impact of various hyper-parameters in DeepLayout on its performance.
Chapter 6

Critter: Augmenting creative work with dynamic checklists, automated quality assurance, and contextual reviewer feedback

The work described in this chapter is based on the following work published in CHI 2019.


6.1 Motivation

Flud (Chapter 4) and DeepLayout (Chapter 5) are novel solutions for enforcing guidelines that conflict and can be captured quantitatively as a metric. However, they lack the ability to address creative tasks with numerous guidelines that are hard to encode. For example,
to ensure quality in creative work, fields such as design and software engineering gravitate toward best practices such as design guidelines [92, 113] or coding style guides [210]. These best practices are often hard to programmatically check. Written design guidelines, while offering standards that should improve quality, pose their own challenges [142, 203]. For instance, multi-page style guides have inherent usability drawbacks like searchability, conflicts, and obsolescence. Web design is one such domain, where experts have to deliver websites that are aesthetically compliant with customers’ requests while also following common best practices around accessibility and responsivity.

Checklists have played an increasingly important role in complex tasks ranging from the cockpit to the operating theater. Their role in creative tasks like design is less explored. In a needfinding study with expert web designers, we identified designers’ challenges with checklist of design guidelines, automated checker, and reviewer feedback. In this chapter, we present Critter, which addressed these challenges with three components: Dynamic Checklists that progressively disclose guideline complexity with a self-pruning hierarchical view, AutoQA to automate common quality assurance checks, and guideline-specific feedback provided by a reviewer to highlight mistakes as they appear. In an observational study, we found that the more engaged a designer was with Critter, the fewer mistakes they made in following design guidelines. Designers rated the AutoQA and contextual feedback experience highly, and provided feedback on the tradeoffs of the hierarchical Dynamic Checklists. We additionally found that a majority of designers rated the AutoQA experience as excellent and felt that it increased the quality of their work. Finally, we discuss broader implications for supporting complex creative tasks.
6.1.1 A Motivating Example: Website Design

To motivate our work, we present an example based on the first-hand experience of some of the authors who work at B12, a company that works with professional web designers to design websites for their customers. Initially, web designers creating websites for B12 would, upon starting a new project, receive a customer brief. This brief contained semi-structured aesthetic preferences, functional requirements, feedback, and content that a new customer provided by filling in a questionnaire. After reading the brief, designers followed different paths. Some would merely start designing the website. Others would jot down reminders to key customer requests in a note-taking application of their choice. After completing a draft of the website, the designer behavior again varied. Some designers simply submitted the draft website whereas others would go through the list of their reminders to ensure they hadn’t missed a detail.

As a set of best practices emerged, B12 presented designers with a Google Spreadsheet containing over a hundred (currently, 153) distinct design guidelines. These guidelines were carefully curated by senior designers at B12 to serve as a template checklist for any new project. Guidelines included topics such as website structure, copy, imagery, and aesthetics, with major topic areas depicted in Figure 6.1A. In spreadsheet form, the list was presented as a checklist on which designers could mark guidelines as completed.

While the guidelines would change with time, most designers rarely reviewed or checked off the guidelines after their first few projects. A few designers would create a short checklist at the start of each project and consult both the guidelines and their checklist before submitting a draft website for review. However, it was common for many designers to do neither of these. While designers provided expertise and kept key customer requests in their memory, they would still miss explicit customer preferences or B12 design guidelines. This poor adherence
resulted in customer frustration, and researchers at B12 sought a system to better present guidelines, assure quality, and capture feedback.

6.2 Pilot Study: A Traditional Checklist

To understand how web designers currently use checklists, we conducted a pilot study in which we shadowed four web designers at B12. We observed each web designer work on one website while sharing their screen with us and talking aloud about what they were thinking while designing a website. Additionally, we conducted a two-part interview session where we asked the designers some questions before and after the shadowing session. The interview focused on questions such as “What’s the main motivation for not using the current checklist?”, “When do you use automated quality checks?”, “What are your thoughts on the feedback you receive from reviewers?” Each interview and shadow session was video recorded and lasted approximately 3–4 hours.

6.2.1 Challenges and Drawbacks

Our pilot study uncovered some drawbacks of using traditional checklists like the flat spreadsheet-based version discussed in Section 6.1.1. We saw that each project was unique and had specific requirements that must be noted and adhered to even if they conflicted with some of the design guidelines. For example, “I’m a sole proprietor, but use ‘we’ in the copy” was a requirement specific to a customer which conflicts with the design guideline “If the customer is a sole proprietor, do not use ‘we’ in the copy.” Existing checklists cannot address such diverse customer needs. We saw that projects also vary in complexity. For example, some customers wanted the designers to focus only on the content, while others wanted bespoke
aesthetics for a multi-page website.

As with projects, designers also had diverse workflows. Traditional checklists, on the other hand, imply highly constrained workflows that did not always align with a designer’s workflow. Indeed, sometimes a designer may have different workflows for different projects due to project-specific requirements. For example, one designer said, “I will focus on aesthetics first since the customer was particular about them.” Overall, designers in the pilot study found the traditional checklist to be lacking the dynamicity to address the diverse project and designer needs.

We also observed challenges unrelated to dynamicity. We learned that designers who didn’t like to use the provided checklist often overestimated their capacity to remember customer-specific details and internalize the design guidelines. These designers also acknowledged forgetting a few design guidelines.

We also found that beyond technology, designers most desired and appreciated reviewer or peer feedback. Designers reported that they found feedback useful because it helped them learn from their mistakes. However, they also felt that it could be frustrating if the reviewers do not review the project requirements before providing feedback. When asked about automated feedback systems, designers reported that an early version of AutoQA was unnecessary and raised a lot of unreasonable errors. Moreover, the designers strongly suggested that they favored human input on their work above generic design guidelines or algorithmic QA.

We distill these findings from the pilot study into the following challenges:

**C1:** Each project has specific requirements that must be noted and adhered to.

**C2:** Designers vary the order in which they complete checklist depending on customer priorities and their areas of expertise.
C3: Only a subset of the design guidelines apply to a project.
C4: Designers overestimate their capacity to remember project-specific details and design guidelines.
C5: Designers desire and appreciate high-quality human feedback.

6.3 Critter System Description

We built Critter with the aim of addressing the challenges (C1–C5) presented in Section 6.2.1. We now provide details on Dynamic Checklists, AutoQA, and the contextual reviewer-provided feedback experience in Critter, highlighting how each component addresses the five challenges, along with the implementation details.

6.3.1 Dynamic Checklists

Dynamic Checklists allow experts to take notes, utilize self-pruning templates, and prioritize their focus on the design guidelines that matter most for a given project. We built Dynamic Checklists as an extension of the open source Orchestra project, a platform for managing flash teams [158] collaborating on creative and analytical projects.

Interface details

The Dynamic Checklists interface, depicted in Figure 6.1A, allows experts to manage various checklist items called todos. At the top of the interface, experts can add todos. An expert can add individual todos related to this specific project by writing and adding a todo with the “Add todo” button (addresses C1). Experts can also add todos from a template by clicking “Add todos from a template.” A template includes all related design guidelines and
Figure 6.1: Critter components: Dynamic Checklists, AutoQA, & Checklist item-specific feedback. HTML and CSS have been edited slightly for presentation. (A) Dynamic Checklists interface (B) AutoQA interface, which provides warnings about potential issues in the context of the B12 website building experience. (C) Checklist item-specific feedback. This feedback is displayed for a current project, and also on checklist items that received feedback on recent relevant projects. (D) Reviewer interface, which allows the reviewers to provide checklist item-specific feedback.

can be applied to any project. In Figure 6.1A, one template called “Launch design checklist” is relevant to this project.

A left column contains todos the expert is responsible for completing. In Figure 6.1A, a designer has added a “Migrate photos...” todo to the top of their list. They have also added the “Launch design checklist,” a collection of 153 nested todos that cover new website design guidelines. As an expert works through a dynamic checklist, they can check items off in any order, or skip irrelevant checklist items. Checklists are hierarchical, so experts can also choose to drill down to any level of specificity. For example, experts seasoned in information architecture can read less detail on “Website structure” while focusing on “Content” (addresses C2).
Prunable and self-pruning checklist templates

Checklist templates like the "Launch design checklist" in Figure 6.1A represent all design guidelines, but only a part of this template may apply to a given project. Dynamic Checklists offer a mixed-initiative system for pruning the irrelevant todos from these checklist templates (addresses C3).

First, the Dynamic Checklist analyzes customers’ request and automatically skips irrelevant todos and their children. For example, the design guidelines cover for both single-page and multi-page websites. If the customer specifically requests a multi-page website, the system automatically skips irrelevant todos that are specific a single-page website project. In Figure 6.1A, the “Apply multi-page...” todos remain in the left column, whereas the “Apply single-page...” todos have been skipped automatically based on customer requirement.

Second, after a designer adds a checklist template, they can further identify irrelevant todos that weren’t automatically detected as “[not relevant]”, moving a todo and its children to the right ”Skipped todo items” column. This mixed-initiative approach aims to reduce a users’ mental load so that they can focus on tasks that require human judgment [228].

6.3.2 AutoQA

AutoQA (Figure 6.1B) is an interface embedded in B12’s browser-based website editor. At any time, a designer can run AutoQA on their work, receiving a report of the errors it can automatically detect in their project. The interface allows experts to chart their path through a project while identifying common aspects of the design guidelines for which experts overestimated their expertise (addresses C4).

Unlike Dynamic Checklists, which was built as a generic add-on to the open source Orches-
tra platform and can be applied to any creative or analytical work, AutoQA was built in the context of B12-specific design projects. Since automation requires context specific to websites, and fixing AutoQA-identified errors requires modifications to customer websites, we embedded AutoQA directly in the website editing experience. When a designer runs AutoQA, a collection of AutoQA checkers (Python functions) analyze the intermediate representation of a website. These checkers scan the content, aesthetics, and website structure for issues such as broken links, visual inconsistencies, information architecture issues, copy and imagery issues, and search engine optimization opportunities.

Once the AutoQA checker identifies these potential issues, they are serialized to the AutoQA frontend as shown in Figure 6.1B. A designer is presented with the AutoQA issues (10 in the figure) and, upon clicking on an issue, is taken directly to the website editor screen in which they can address the issue. The design guidelines instruct designers to address AutoQA issues during a “Website QA” phase, thus connecting AutoQA and Dynamic Checklists within the Critter environment.

### 6.3.3 Contextual Reviewer-provided Feedback

The Dynamic Checklists interface allows reviewers to provide human feedback on other experts’ projects. The design guidelines serve as a rubric for feedback [225], with each piece of unstructured feedback (e.g., “The Gallery hero image on the home …”) linked to the relevant aspect of the checklist hierarchy (e.g., “Make sure the website looks good responsively on mobile and tablet views”). The interface offers reviewers a structured environment to provide the feedback for which experts expressed a preference (addresses C5) as shown in Figure 6.1D.

To promote a two-way communication via feedback, Critter allows the reviewers to easily copy and send this feedback in markdown format along with a hyperlink to the Dynamic
Checklist with item-specific feedback as depicted in Figure 6.1C. To further facilitate learning, the interface also displays the feedback an expert received on previous projects as a recommended todo. These recommended todos are highlighted in red along with the feedback (see Figure 6.1C). This allows an expert to keep an eye out for the mistakes they tend to make in practice.

In aggregate, AutoQA and contextual reviewer-provided feedback serve a more holistic purpose [57, 149]. When experts repeatedly encounter the same set of AutoQA errors or human feedback in particular areas of a checklist, the encounters serve as a reminder for experts to dedicate more attention to these areas in future projects.

### 6.4 Evaluation

Our evaluation of Critter addressed the following research questions:

**Q1** How did designers’ use of Critter affect the quality of their designs?

**Q2** How did designers feel about using Critter?
Task

We selected website design as the creative task under study. This task aligns with our motivating example while also encompassing a range of technical and aesthetic tasks characteristic of many creative tasks.

Participants

We recruited six designers who regularly design websites for B12’s customers. All of the designers were trained to use the B12 website editor and Critter. Some were also part of the initial pilot study that helped us identify the challenges for Critter described in Section 6.2.1.

Procedure

We asked each designer to design five websites for actual B12 customers. The customers were generally small- and medium-sized businesses (SMBs), and their requirements generally ranged from 2–7 hours of design work, optionally requiring copywriting. For each website, the designers created an initial checklist using the “launch design checklist” shown in Figure 6.1A. The designers were allowed to add project-specific custom checklist items in addition to the templated items. After creating the checklist, the designers were instructed to use Critter as they saw fit. At the end of the design task, we asked designers to complete the checklist item that asks them to run AutoQA and resolve all of the reported errors.

After a designer completed a website, an expert reviewed their work and provided feedback via the Critter interface within 30–40 minutes. This turnaround was fast enough to allow the designer to consider the feedback before starting the next website in all but one case. The reviewer was an author of the paper associated with this chapter with four years of
experience in web development and two years of experience in user experience (UX) design. The reviewer evaluated the websites based on their adherence to the 153-item template checklist and provided structured feedback as described in Section 6.3.3.

After completing all the websites, designers completed an online survey that asked them to rate their experience with Critter on several five-point Likert scales. The survey also asked some open-ended questions about how they used Critter and how it impacted the quality of their work. We used the survey responses as the basis for follow-up interviews that we conducted with each designer over video calls. All of the interviews were audio-recorded and partially transcribed based on detailed notes. One of the designers opted out of the final user interview and the online survey. We thus present our results based on the 30 websites delivered by six designers, but our final survey and interview results are based on five of the six designers’ responses.

In addition to the final websites, interviews, and survey results, we also instrumented Critter to record user interactions and performed logs analysis as described in Section 6.5. Finally, designers provided partial or complete screen recordings for 27 of the 30 website design sessions, which helped us learn how designers were using Critter. A preliminary analysis of these recordings suggested that the designers spent a varying amount of time interacting with different UI elements in the system. To further investigate designer’s use of Critter and its impact, we analyze the available data in the following section.

6.4.1 Limitations

One of the limitations of our study is that we do not have quantitative data on the control condition: how did designers perform with traditional flat checklists? This limits our ability to make broader claims about the use of traditional checklists and their impact on quality
of work in creative tasks. However, our pilot study helped us uncover challenges of using a traditional checklist in the context of creative tasks, and made it clear that adherence and understanding of the traditional checklist was low.

Another limitation of our work is the study sample size. Because we studied six professional web designers as they created 30 websites for B12’s customers, we were limited to a comparative study between different designers. A larger sample would allow us to perform more robust quantitative analysis and make stronger claims to generalization. However, the rich performance data on diverse real-world websites, interviews, surveys, and instrumented interaction logs enabled us to triangulate our claims across multiple data sources.

6.5 Q1: How did designers’ use of Critter affect the quality of their designs?

We use the number of design guidelines marked as missed by the reviewer as a proxy for the quality of a designer’s work. The lower the number of missed design guidelines, the higher the website quality. We report the number of mistakes per designer in Figure 6.2A, and sort each of the subfigures in Figure 6.2 by the mean number of mistakes per designer across the websites they designed for the study.

The results in Figure 6.2 suggest that despite similar instructions, different designers designed websites of varied quality. Across several measures of engagement with Critter (Subfigures C–G), we found that the more engaged a designer is with Dynamic Checklists, AutoQA, and reviewer feedback, the fewer mistakes they made in following design guidelines. Figure 6.2B, which provides a count of the number of websites designed by each designer for B12 prior to the study, suggests that novice designers who adhered to the system made fewer mistakes
than experienced designers who didn’t.

We next explore how engagement with each component of Critter —Dynamic Checklists, AutoQA, and reviewer feedback—affect[ed participants’ website quality.

6.5.1 Dynamic Checklists

Checking items off

Marking a checklist item as complete is the most essential interaction in Dynamic Checklists. We asked all of the designers to only check off items they have read and addressed while designing the website.

In Figure 6.2C, we note that the designers with fewer checked off items generally made more mistakes, whereas designers with more checked off items generally made fewer mistakes. D1, who checked off the second largest number of items and had the lowest average mistakes per task said, “I like that it guides me through the whole process, making sure that I don’t forget anything.”

We noticed that 61% of the missed guidelines were marked as complete by the designers. In Figure 6.3, we see that the majority of these mistakes were made by designers who had larger checked-off item counts (Figure 6.2C). When asked about this oversight during the interview, one participant (D5) mentioned having “limited time,” and identified situations in which it was “very hard to pay attention,” especially to design guidelines that were missed rarely.

In contrast, designers with fewer completed items not only missed some completed items, but also missed a lot of incomplete items, as shown in Figure 6.3. Some designers explained that they felt that they were experienced enough to have the steps memorized and did not
Figure 6.3: Bar plot of the average number of missed incomplete/complete design guidelines. The y-axis represents the designers arranged in increasing order (top to bottom) of the mean number of mistakes made by each expert.

feel the need for explicitly checking items off. In practice, these designers appear to have overestimated their capability to remember design guidelines for each task.

Skipping

Skipping checklist items is one of the core operations which allows Critter users to prioritize the design guidelines that are relevant to the task. To measure the level of user’s engagement with the skipping functionality, we compute the ratio of the number of checklist items marked as irrelevant to the size of their completed checklist.

In Figure 6.2D, we see that designers who skipped fewer items generally made more mistakes than designers who skipped more items. We note that 0.6% of mistakes were on items that designers skipped as irrelevant. This shows that designers were successfully able to prune irrelevant items, reducing their chance of missing an item due to an oversight.
6.5. Q1: How did designers’ use of Critter affect the quality of their designs?

**Toggling focus**

The accordion-style hierarchical checklist allows designers to switch and compartmentalize their focus on a subset of checklist items at a time. To estimate participants’ engagement with this functionality, we measure the number of times a participant toggled (expanded or collapsed) a checklist item.

The results in Figure 6.2E indicate that participants who used the toggling functionality the most also delivered the highest-quality websites. Asked to explain their toggling behavior, one participant (D1) said, “It was convenient because it was logically divided and allows the designer to open up one branch and close it when it is done.”

**Drilling deep**

The hierarchical nature of Dynamic Checklists allow users to explore and drill down to the level that fits their design process. We estimate the level of drilling interaction by the median depth of the checked off items in the checklist.

While not plotted due to space constraints, designers who drilled deep generally made fewer mistakes than the designers who only used higher-level design guideline topics.

6.5.2 AutoQA

AutoQA provides near instantaneous feedback to participants. In Figure 6.2F, we show the percentage of tasks in which participants eliminated all AutoQA-reported errors, a measure of engagement with AutoQA.

We note two observations from this data. First, the AutoQA functionality had relatively high usage among participants, with all participants eliminating all AutoQA-reported errors.
on the majority of their projects. Second, we notice a slight trend in which experts with higher AutoQA engagement made fewer mistakes.

6.5.3 Reviewer Feedback

As explained in Section 6.3.3, recommendations from reviewers allow designers to keep an eye out for mistakes on previous tasks. When a designer deviates from a guideline, the reviewer points out the mistake in the feedback and the guideline gets highlighted on the designer’s next project. In Figure 6.2G, we measure the percentage of recommendations that participants adopted in their next project.

We observed strong adherence to the recommended todos: 88 out of 91 recommendations across 30 websites were successfully applied. In three cases where participants repeated a mistake, the designers explicitly differed from and pushed back on reviewer judgment. Due to the strong adherence to recommendations across all participants, we were unable to observe its effect on mistakes based on engagement.

6.5.4 Usage Model

In The Checklist Manifesto [70], Gawande discusses two checklist usage patterns: 1) do-confirm, where a user can perform (do) one or many tasks and check off (confirm) the corresponding checklist items; and 2) read-do, where a user reads (read) a checklist item and performs (do) the corresponding task. We found two primary usage patterns for Dynamic Checklists amongst participants. The primary pattern is a hybrid of the two usage models: do-confirm-read-skip-do. In this model, the designers finished (do) majority of the tasks, checked (confirm) corresponding checklist items, went through (read) the list of remaining incomplete items, actively skipped irrelevant items, and finished (do) relevant checklist
items. The second pattern of *read-do* manifested in two newer experts to web design with B12 (D1 and D4). Early in their experience with checklists, they serially *read* the guidelines and completed them (*do*). As they gained confidence with the guidelines, they switched to *do-confirm-read-skip-do*. The designers did not explicitly identify these models and there is no evidence to suggest that either of the two models is better than the other in terms of the designer’s quality of work. The designers reported that they chose to switch to *do-confirm-read-skip-do* model because it allowed them to design and cover majority of the checklist items from their working memory without switching focus to the checklist.

### 6.6 Q2: How did designers feel about using Critter?

![Bar plot visualization](image)

*Figure 6.4: Bar plot visualization of the average rating given by the designers on a five-point scale where 1 represents poor and 5 represents excellent user experience.*
6.6.1 Overall Experience

As depicted in Figure 6.4, Critter received a moderately good overall experience rating of 3.4 on a 5-point scale where 5 is an excellent experience and 1 is a poor experience. D2 summarized, “It is a helpful reminder of things that can easily get lost in all that has to be done to build a website.”

6.6.2 Dynamic Checklists

Designers reported that they liked the idea of automatically pruning irrelevant checklist items from their checklist (mean rating = 3.4) as it saves time. As described in Section 6.3.1, Critter automatically pruned 20.44% of checklist items for each project on average (std dev 5.07). D5 explained, “I thought it was smart and worked well. In one project I reviewed these automatically skipped items just in case, and I don’t think there was anything there that should have been included.” Other designers described not noticing automatically pruned items, with D6 saying, “You are not meant to notice it anyway.” These comments support our design goal to move irrelevant checklist items out of focus.

Some designers reported that they liked the hierarchical presentation of checklist items because it allowed them to compartmentalize their focus. D1 explained, “I think they are logical and help concentrate on one aspect of the website at a time. It was convenient because it was logically divided and allows the designer to open up one branch and close it when it is done.” They found the hierarchical format easier to use than the original flat list of checklist items, which were perceived as hard to navigate through and overwhelming.

However, other designers were less enthusiastic about the hierarchical checklists. One designer (D6) felt that it did not fit into their design flow and felt like a chore. D5 maintained their stance that, as an experienced designer, they did not need to use the thorough checklist
and spend time on moving out irrelevant items from their checklist. Instead, D5 expressed the desire to use a reusable shorter cheatsheet checklist that they can update based on reviewer feedback. Furthermore, the designers unanimously reported readability issues with hierarchical checklists and therefore rated it poorly in comparison to other components (mean=2.8). They suggested different fonts, borders, or color schemes to more clearly delineate different sections of the hierarchical view.

6.6.3 AutoQA

Four out of five designers rated the AutoQA experience as excellent (mean rating = 4.6) and felt that it improved the quality of their work. Designers reported two key benefits of using AutoQA. First, they felt it helped them catch errors they would miss otherwise. D4 noted, “I look forward to it each time. It makes me accountable.” Second, they felt AutoQA also acted a learning tool for understanding B12’s design guidelines. In D1’s words: “Now I understand what kind of mistakes I can make. It helped me learn more about my design process. Now I try to make sure I don’t repeat AutoQA errors before running AutoQA.”

6.6.4 Reviewer Feedback and Recommendations

All of the designers expressed appreciation for the human reviewer’s feedback (mean rating = 4.4) as it helped them learn about B12’s expectations and improve their design process. D1 said, “It actually helped me to better understand some todo items and and what results are expected.” Designers also liked the fact that the feedback appeared on their checklist, helping them to understand the approach the reviewer took to give feedback. Given the hierarchical nature of the checklist, the feedback next to the checklist item not only helped them identify what needed to fixed, but also the general area in their design process that
needs attention.

Designers also scored the automatically recommended todo items from previous projects highly in the survey (mean = 4.0), but were more reserved in their interview comments. Most designers did not feel that the recommendations directly impacted the quality of their work. However, all of them acknowledged that highlighting the todos in red and attaching reviewer feedback caught their attention. D1 felt that it indirectly prevented mistakes, saying, “It makes me pay more attention to the particular todo items so that I don’t make the same mistakes again”. Some designers also acknowledged its potential as a learning tool. D5 said, “It would be an issue you were struggling with and by showing up, it reminds you to work on it”.

6.7 Discussion and Conclusion

We now discuss some practical implications for systems that aim to augment creative work and speak to the generalizability of Critter.

6.7.1 Implications for Practice

A diversity of approaches

Systems that aim to support high-quality creative work should include more than one way to ensure the quality. To handle the different forms of mistakes experts might make across projects, Critter uses complementary quality assurance techniques like Dynamic Checklists and AutoQA to eliminate these mistakes. The various components of Critter address various scalability issues with feedback. At one extreme, AutoQA and automatic pruning offer instantaneous feedback and review scalability, but are limited to computable guidelines and
requires subject matter-specific automation. At the other extreme, Dynamic Checklists that embed reviewer feedback cover the long tail of guidelines and quality assurance without the implementation challenges, but lack the immediacy of automation and are susceptible to errors inherent to human judgment [182].

**Automated critics are effective when accurate**

Corroborating prior work [70], automated critics like AutoQA should only report high-confidence errors transparently identifying doubt in errors. In early prototypes, experts were critical of automated feedback. Whereas participants trusted and appreciated reviewer feedback, they quickly grew suspicious of automated errors with high false positives. After we reduced the AutoQA false positive rate, participants’ opinion of AutoQA changed, and the majority rated their experience with it as excellent.

**Design guidelines as rubrics lead to learning and better work**

We observed that some experts expressed a better understanding of the review process by using AutoQA. Experts also felt more confident going through the checklist after resolving the guidelines-specific errors reported by AutoQA. Similarly, we observed that experts were able to self-reflect on their design process by going through guideline-specific feedback. These observations are in line with Dow *et al.*’s finding that using specific criteria as self-assessment rubrics leads to learning [43]. That said, we also note that factors like experience and motivation [171] also play an important role in designers’ ability to learn from self-reflection.
Hierarchical checklists support task non-linearity and varied expertise

The hierarchy of Dynamic Checklists affords both a broad and deep exploration that is ideal for non-linear tasks like writing, poster design, or filmmaking. Hierarchical checklist organization also allows experts to stay at a different level of granularity than novices. During our pilot study, we learned that web designers did not like a traditional flat checklist because it did not align with their process. In larger projects, they found themselves scrolling the checklist multiple times to revisit different parts of the checklist to explore various subtasks. A hierarchical view facilitated a quick scan with deeper exploration into the relevant area of focus.

6.7.2 Alternative Design Opportunities

We now discuss a key design choice in Dynamic Checklists. Critter asks experts to build a checklist by pruning the template checklist down to relevant items in a top-down fashion. An alternative design could allow the experts to build their checklists from scratch by adding relevant items in a bottom-up fashion instead of skipping them. We selected a top-down approach to enable experts with less experience in the guidelines to explore and internalize them with time. For experts who have already internalized most of the checklist and do not want to spend time on removing irrelevant items, a bottom-up approach that enables them to add relevant items might fit their workflows better. Designing a dynamic system that supports such interactions while ensuring the adherence to the guidelines is an interesting future direction.
6.7. Discussion and Conclusion

6.7.3 Generalization to other Domains

We now argue that use of self-pruning hierarchical Dynamic Checklists, AutoQA, and context-specific feedback can augment the quality of work in other areas which require expertise like writing, movie-making, and software engineering.

At their core, the design guidelines, automated checks, and reviewer feedback from our motivating example have analogies in many creative fields. The web design guidelines in this chapter are akin to written style guides [65, 179] (e.g., “use the oxford comma”) or programming style guides [1, 210] (e.g., “Class names should be nouns in UpperCamelCase”). Similarly, automated systems like Grammarly [2] for writing or linters [103] for programming play a similar role to AutoQA. Finally, reviewer feedback can take the form of an editor for writing or code review for programming.

With these analogs in mind, one can imagine generalizing Critter to other creative fields. For example, before submitting a code review, a software engineer might be presented with a hierarchically organized style guide for their programming language that has been pruned to relevant aspects of the task, checking off key areas they have completed. If the engineer has received code review feedback on certain aspects of the style guide in the past, those areas can be recommended for deeper inspection. Before submitting code for review, the engineer must ensure their code free of linter errors. Finally, a code review can enforce quality while informing future tasks. As the programmer internalizes the checklist, they can inspect it at coarser levels of granularity.

With Critter, we have demonstrated that creativity can be augmented by structure. While this work warrants more exploration, we believe Dynamic Checklists, AutoQA, and contextual reviewer feedback to be part of a brighter future of work.
6.8 Chapter Summary

In this chapter, we argue that Critter augments creative work through Dynamic Checklists, AutoQA, and contextual reviewer feedback. To evaluate the effectiveness of Critter, we conducted an observational study with professional web designers creating websites for 30 real-world customers at B12. We found that Critter was able to effectively incorporate design guidelines into a designer’s process and consequently generate higher quality websites. We showed that the more engaged a designer is with Critter’s self-pruning hierarchical checklists and AutoQA, the fewer mistakes they make in following detailed design guidelines. We also found that novice designers who adhered to the system made fewer mistakes than experienced designers who didn’t. Survey and interview data suggest that the designers generally liked Critter (mean rating=3.4) and found it to be a useful guidance tool. Based on our findings, we also discussed broader implications of our research to complex creative tasks in other domains.

In summary, the contributions of this chapter include:

- A novel approach, informed by a pilot study highlighting gaps in traditional checklists, that combines automation with human-driven planning and reflection to help experts navigate complex creative tasks. Specifically, we augment traditional checklists with a) a hierarchical structure to mask complexity and offer navigability, b) automation to prune and instantaneously check guidelines whenever possible, and c) self-reflection through guideline-specific feedback.
- A system, Critter, that demonstrates this approach with three software components: a) Dynamic Checklists that are hierarchical and self-pruning, b) AutoQA to instantly identify common mistakes, and c) Contextually presented reviewer feedback.
- A mixed-methods evaluation of web designers as they create 30 websites for B12 using
Critter. Through this study, we find that the more engaged a participant is with Dynamic Checklists and AutoQA, the fewer mistakes they make in following design guidelines. Participants also rate the three components of Critter highly, and provide useful feedback on the tradeoffs in a hierarchical display.
Chapter 7

Conclusions

In this chapter, I conclude my dissertation and discuss future work. First, I revisit the main research questions listed in Chapter 1 and discuss how the research presented in my dissertation addresses each of them. Next, I summarize the broader implications of the research discussed in this dissertation. Finally, I discuss potential opportunities for future work in method development for following design guidelines in creative tasks.

7.1 Addressing the Research Questions

7.1.1 RQ1: How can we design a single system with the ability to store, organize, share, explore, and lay out graph visualizations?

The GraphSpace web platform presented in Chapter 3 addressed Research Question 1 by providing a rich set of user-friendly features that enable users to store, organize, share, explore and lay out graphs. Specifically, GraphSpace allows users to store and access richly-annotated networks and their layouts online. It also includes a powerful graph drawing interface which allows users to edit node and edge styles, lay out networks, and save multiple layouts for a graph. The platform also allows users to create private groups to organize project specific networks as a collection. Users can also add their collaborators to the group
by their email addresses, and share networks with the entire group. Group owners can also ask new users to join a group with an invitation link. GraphSpace provides a rich search functionality where users can explore networks with specific annotation(s), node(s), and/or edge(s). GraphSpace also gives its users an option to make their network(s) public, enabling others to access these networks via a persistent URL.

GraphSpace was started as an online tool to facilitate a collaboration on budding yeast networks involving cell cycle regulation \[122, 150\]. Since then, GraphSpace has found a wider use in network biology projects and assists in accelerating all aspects of collaborations among computational biologists and experimentalists, including preliminary investigations, manuscript development, and dissemination of research. Currently, GraphSpace supports around 500 users who have stored 25,966 graphs (most of them private) with a total of over 2 million nodes and 5.4 million edges. The users have organized their graphs in 198 groups and created around 11,000 layouts (most of them shared privately). Since its release, the GraphSpace open source community has actively contributed to the project codebase and added new features like a programmable graph legend interface, graph forking, and new layout options like Cola. The open source project also has pull requests for features like a versioning system, comments/discussion system, and PDF export which will further enhance the capabilities of GraphSpace as a collaborative platform.

GraphSpace’s features have enabled researchers to perform network-based collaboration from any part of the world on a single platform. It serves as a bridge between visualization and analysis of individual networks supported by systems such as Cytoscape \([189]\) and the network indexing capabilities of NDex \([151]\). Most importantly for this dissertation, GraphSpace plays a critical role in the success of my subsequent research (Chapters 4 and 5) in the graph visualization domain. It provided the core infrastructure needed to store, organize, share, and analyze graphs and layouts created by Flud (Chapter 4) and DeepLayout (Chapter 5).
7.1.2 RQ2: How can we use a mixed-initiative approach to enforce design guidelines in graph visualizations?

After setting up the core infrastructure (GraphSpace) that supports network-based collaboration, I investigated the use of mixed-initiative methods to enforce complex and conflicting design guidelines in graph visualization domain as part of Research Question 2. In Chapter 4, I specifically focus on four previously defined aesthetic considerations, plus a new biologically inspired criterion — “maximize the number of downward pointing paths” — for a network drawing task. These guidelines can be encoded in software and it is possible to measure quantitatively how well a layout adheres to them.

To address the challenges with satisfying conflicting quantitative and complex guidelines, I proposed a novel hybrid crowd-algorithm strategy for graph layout. It is a sequential, collaborative process wherein crowd workers and a Simulated Annealing-based layout algorithm [38] build on each other’s progress. Here, I leveraged the creativity and human intelligence of crowds to scale up the guideline-specific graph drawing tasks (also called task modes in Chapter 4). I further utilized the fine-tuning capabilities of Simulated Annealing algorithm that complemented the crowd’s ability to solve complex scenarios. To facilitate the hybrid crowd-algorithm collaborative process, I implemented a system called Flud. Flud manages the crowd worker recruitment and hand-off between crowd workers and Simulated Annealing-based layout algorithm. Flud also facilitates a layout interface for crowd workers. This interface gamifies the complex domain-specific graph drawing task and allows novice crowds to draw biologically meaningful graph visualizations with the help of algorithmically-generated suggestions.

Flud employs two key features in its layout interface to augment crowds’ capabilities. First, Flud provides crowd workers with criterion-specific priority values and scores. Flud asks
workers to make modifications to the layout so as to balance the criterion-specific scores based on the direction of flow of information in the network, edge crossings, and relative distances between nodes and edges. These features enable workers to track their progress and help them to focus more on one constraint than another in the case of a conflict. Second, Flud provides workers with a “clue” for each criterion. If a crowd worker is stuck, perhaps because the network is complex or it is unclear what the next move should be, the system highlights a small subset of nodes and edges in the network as clues. These nodes and edges are selected by Flud such that changing the positions of these elements is likely to improve the score for the corresponding criterion. Overall, this feature focuses the crowd worker’s attention on a smaller, more tractable problem, allowing crowd workers to succeed at the given task.

To evaluate Flud and its novel mixed-initiative layout approach, I conducted a two-part empirical study on Amazon Mechanical Turk with nearly 2,000 novice crowd workers. In this study, I asked the crowd workers to lay out and visualize complex protein networks that represent signaling pathways in human cells as part of a game. The first part showed that assigning the task modes in the order of their priorities generates better results than random assignment. The second part of the study showed that the collaboration between humans and algorithms leads to higher scoring layouts than either from humans or algorithms alone. In another experiment with participants who have biological training, I provide empirical evidence showing that participants were more accurate, faster, and more confident while performing these tasks with Flud-generated layouts in comparison to algorithmic baselines. Overall, the results indicate that layouts generated via mixed-initiative approach are of higher quality, more meaningful, and easier to understand than the ones generated by algorithms.

While the hybrid crowd–algorithm approach used in Flud offers a reasonable solution for enforcing conflicting and complex domain-specific guidelines, the computer agent (i.e., Sim-
ulated Annealing) in the approach lacks a learning capability. The algorithm only uses the layout generated by the crowd workers as input and disregards how the crowd workers created the layout. Moreover, the algorithm does not learn from its past experience. Due to the lack of learning ability, the algorithm solves similar sub-problems inefficiently in each run. Moreover, according to Ferguson and Allen [55], for a mixed-initiative system to be truly collaborative, the computer agent in a mixed-initiative system should have the capability to learn new ways to perform a task. All of these reasons motivated me to design a learning-based method as an alternative to replace Simulated Annealing in Flud’s hybrid approach.

### 7.1.3 RQ3: How can we design a computer agent with the ability to learn how to enforce design guidelines in graph visualizations?

To address Research Question 3, I present a learning-based approach to graph visualization in Chapter 5. Specifically, I propose a reinforcement learning technique, DeepLayout, to generate a graph drawing that optimizes a given layout quality function. In DeepLayout, given a set of structurally similar graphs as input, I train a model to improve the layouts of these graphs iteratively. DeepLayout addresses the above-mentioned limitation seen in Simulated Annealing by learning an implicit representation of the layouts and a mapping from layouts to actions with the long-term goal in mind. These actions iteratively update the given layout by selecting a node and moving it to adjacent grid cells in a discretized layout space. Later, given a new graph that is structurally similar to the input set, I use the trained model to generate a drawing through an iterative layout process. In this layout process, I start with a random layout or one created by an existing algorithm. Then, I iteratively use
the trained model to convert the layout to its implicit representation, generate an action corresponding to this representation, execute the action, and generate a new layout.

To the best of my knowledge, DeepLayout is the only graph layout approach with the ability to learn from a layout process to generate drawings for new, unseen graphs. I use DeepLayout to learn the layout strategy for efficiently optimizing the layout score/energy function used in Flud. To showcase its utility, I present an empirical study that evaluates DeepLayout’s effectiveness and efficiency in drawing new graphs that are structurally similar to the training data. The evaluation was conducted for 12 synthetic graph datasets. The graphs in each dataset were structurally similar to each other and generated from directed graphs representing signaling pathways in human cells. The results show that, on average, DeepLayout outperforms state-of-the-art methods like Dig-Cola and IPSEP-Cola in all datasets, and the Simulated Annealing method in nine (75%) of the datasets. Furthermore, I show that DeepLayout was, on average, 23.6% faster than the Simulated Annealing method. Finally, I use qualitative comparisons to demonstrate that the layouts created by DeepLayout are superior to those output by a Simulated Annealing-based approach and other state-of-the-art algorithms designed for drawing directed graphs.

In real-world applications, this learning-based approach can benefit novices with no expertise in machine learning or graph visualization. For instance, the model can be trained with cost functions designed by experts to capture domain-specific needs and aesthetic requirements. Later, the novices can directly use these well-trained models to lay out their graphs.
Chapter 7. Conclusions

7.1.4 RQ4: How can we use mixed-initiative approach to enforce design guidelines in web design task?

Finally, as part of Research Question 4, I study the web design task because, unlike graph visualization, the number of guidelines (usually > 50) and design elements in this task are typically large and it is mostly hard to programmatically check if a guidelines has been satisfied. Traditional quality assurance techniques like checklists have been effective in facilitating the fulfillment of guidelines, that are hard to programmatically check, in domains with semi-structured tasks [70]. While review and manual quality assurance help improve quality, they are expensive and may not lead to improvement in expertise. That said, at the time, it was unclear how we can use traditional quality assurance techniques like checklists, automated critics, and expert feedback for creative tasks. Therefore, I investigated the iterative and real-world deployment of these quality assurance techniques for enforcing design guidelines with a team of professional web designers. As part of this research, I conducted a needfinding study with expert web designers that uncovered five challenges (or requirements) for a new system. First, each project has specific requirements that must be noted and adhered to. Second, designers vary the order in which they complete a checklist depending on customer priorities and their areas of expertise. Third, only a subset of the design guidelines apply to a project. Fourth, designers overestimate their capacity to remember project-specific details and design guidelines. Finally, designers appreciate high-quality human feedback.

Informed by these needs, I created Critter, a mixed-initiative system that helps experts efficiently create effective checklists that dynamically adapt to individual project requirements. Critter has three key components: Dynamic Checklists, AutoQA, and contextual reviewer feedback. Dynamic Checklists augment checklists with a hierarchical structure to mask complexity and offer navigability, and an automation to prune and promptly check guidelines
when possible. AutoQA is a critique system that performs an automated quality assurance check for common errors identified in the checklist. Finally, Critter allows reviewers to provide feedback around specific design guidelines that are highlighted in Dynamic Checklists on future projects to promote learning and iterative improvement.

To evaluate Critter’s effectiveness, I conducted an observational study with professional web designers creating websites for 30 real-world clients at B12, a web design firm. In the study, designers used Dynamic Checklists and AutoQA as part of their process while receiving contextual feedback from B12 employees. During this study, I quantitatively observed the interaction of designers with the system. I found that the more engaged a participant was with Dynamic Checklists and AutoQA, the fewer mistakes they made in following design guidelines. I also found that a majority of designers rated the AutoQA experience as excellent and felt that it increased the quality of their work. Overall, my findings revealed that with mixed-initiative interactions, checklists can be effectively used in creative domains that have numerous best practices, idiosyncratic client requests, and non-linear completion paths (e.g., designing, writing, programming).

7.2 Broader Implications

I now discuss broader implications of the research work presented in my dissertation for systems that aim to support creative tasks and crowdsourcing research in general.
7.2.1 Implications for Creativity Support Tools

Design guidelines as rubrics to promote learning and better work

In the systems presented in my dissertation, I have used guidelines as rubrics to provide feedback and help the users to self-reflect on their design choices. For example, in Flud, I quantify a design’s adherence to a guideline as a metric and then use these metrics to compute a layout score that measures the quality of the given solution. Flud allows the crowd workers to track the layout score for the current solution and the best solution seen during a game. It helps them to understand where they are with respect to their goal. Furthermore, the goal to beat the current top score and set a higher goal for the other players, gives the crowd workers an incentive to do better work. Flud also displays the per-criterion scores along with the overall score. It helps the crowd workers to understand how their layout modifications are being evaluated. Flud also synchronously indicates the the relative change (increase/decrease) in per-criterion scores after each move. This design choice gives the crowd workers more granular feedback required to self-reflect on how their actions impact the per-criterion scores and the overall quality of the layout. Overall, the Flud’s scoring system, which uses design guidelines as rubrics for feedback, provides a framework for novice crowd workers to learn from their actions and devise a design strategy on the fly to effectively enforce guidelines during a task.

Similarly, Critter allows reviewers to provide feedback around specific design guidelines that are highlighted in Dynamic Checklists on future projects to promote learning and iterative improvement. AutoQA also provided check for common errors identified in the design guidelines and used them to remind the designers of potential mistakes. I found that guideline-specific reviewer feedback and automated quality checks helped designers’ to self-reflect and effectively enforce guidelines in future projects. I also found that by resolving errors reported
by AutoQA in Critter system (which used design guidelines as rubrics), experts felt more
certain going through the checklist of guidelines, with some experts expressing a better
understanding of the process.

These findings corroborate prior work which shows that utilizing specific criteria as a rubric
for self-assessment has a learning effect [43]. Still, guideline-specific feedback can only assist
self-reflection, whereas the designers’ ability to learn from self-reflection also depends upon
other factors like experience and motivation [171]. That said, systems should still consider
leveraging design guidelines as rubrics to boost learning and improve work quality. They set
up a more consistent evaluation system and offer designers a window into the minds of the
reviewers. It will allow the designers to self-reflect and further refine their design process.
At the same, they could also act as a learning tool for novices who are new to a domain and
may not know how to address certain guidelines in a scenario.

Checklist of design guidelines for quality control in creative tasks

Checklists have been increasingly used to perform quality control in semi-structured tasks
with linear, step-by-step completion paths. Unfortunately, performing creative tasks is not
as easy as conforming to a protocol. In a pilot evaluation, I identify the root cause for the
poor adherence to checklists and an effectively lower-quality work product in a web design
task. I found that traditional flat checklists lacked the dynamicity necessary for the diverse
design projects. For example, successful designers can have drastically different workflows,
and many designers might vary their approach on projects of different complexity or scope.
Some aspects of design are also iterative, making the application of step-by-step checklists
more challenging. In Chapter 6, I present a novel checklist design, Dynamic Checklist, that
can be effectively used for creative tasks.
Dynamic Checklists allow experts to create a customized checklist by skipping design guidelines that they consider not relevant to the project or their design process. Related checklist items are grouped and hierarchically nested to progressively disclose details. For any particular project, a checklist is automatically pruned to remove irrelevant guidelines (e.g., single-page website guidelines for a multi-page website). For bespoke requests, experts can use the Dynamic Checklists to add checklist items they will want to complete later in the project.

Using design guidelines as a hierarchical dynamic checklist can offer experts multiple benefits. First, they will not only ensure compliance with critical design guidelines but also the repeatability and completeness of the task. Second, pruning irrelevant guidelines out of focus will allow experts to keep critical items in focus. Third, the ability to toggle parts of the checklist will allow experts to revisit a checklist in non-linear tasks. Finally, it will also allow users at different levels of expertise to use the checklists with different levels of granularity.

On one hand, it allow experts to switch focus between shorter sub-checklists. On the other hand, novice users can use more granular checklists. While further exploration is needed, I believe the hierarchical structure is a promising direction to effectively use checklists in tasks with numerous guidelines.

### 7.2.2 Implications for Crowdsourcing Research

Crowd powered mixed-initiative systems are effective in layout design tasks

Previous work by Singh et al. [185] showed that non-expert crowds can effectively perform a biological graph layout design task if we translate domain knowledge into representational guidelines. However, Singh et al. focused on only natural language guidelines. Flud further extends this work by incorporating concepts from game design into the layout interface,
allowing non-expert crowds to enforce conflicting guidelines that can be quantitatively encoded as metrics. The results of the Flud study replicate the main finding in CrowdLayout, i.e., that non-expert crowds are good at laying out networks, a task that requires observation and spatial arrangement to maintain a certain direction of network flow. My dissertation further builds on CrowdLayout by exploring the potential of hybrid approaches in layout design tasks. Corroborating prior work \cite{7, 33}, I found that combining crowdsourced human intelligence with automated approaches can achieve results better than either approach alone for this layout design task. An implication of this finding is that systems can leverage a crowd-powered mixed-initiative approach to solve for other types of layout design tasks that require careful relative arrangement of entities in a two-dimensional space. Examples of such tasks, which manifest in many creative and analytical disciplines, include drawing integrated circuits, mapping social networks, and designing interior spaces in houses and buildings.

Preference elicitation support enables non-experts to balance multiple goals

Balancing multiple goals in a task can be challenging for novice crowd workers, especially if goals conflict with each other under certain scenarios. Experts know when and how to balance multiple goals to benefit the overall task. However unlike experts, novices are limited by their incomplete knowledge. It is difficult for a non-expert to tell whether one goal is more important than another goal, and to what extent. It may be even more difficult for a user to understand the consequences of choosing to focus on one goal before another. Therefore, preference elicitation support is required to help the non-expert user decide on a certain course of action \cite{6}.

In the context of biological graph layout task introduced in Chapter 4, layout criteria translate to multiple goals for crowd workers. The ability to balance multiple layout criteria (or
goals) is important because they can conflict with each other. A computational biologist who is an expert at creating biological graph visualizations can use his or her domain knowledge to balance multiple layout criteria in case of conflict. However, novice crowd workers who have no expertise in biology and computer science will find it challenging to understand when and how to choose to focus more on one criteria (or goal) than another. Flud reduces this challenge by transparently showing the per-criterion scores, their priorities, and the relative change (increase/decrease) after each move. These features allows the workers to take informative decisions while trying to balance multiple criteria. The Flud study showed that novice crowd workers were able to effectively balance multiple layout criteria. Overall, these results from my work demonstrate that crowdsourcing is a promising solution to scale complex tasks with multiple conflicting goals.

**Computationally assign task modes to break workers out of unproductive task behavior while balancing multiple goals**

As mentioned earlier in Chapter 2, task modes are often used by games to focus players attention on one sub-goal at a time. Task modes are common in human computation games like Phlyo [112] and Foldit [33] where players are allowed to select a mode for a given scenario. These systems facilitate a collaboration between humans and computers with an assumption that novice players can identify which task mode is required for the given scenario. However, in my dissertation research I observed that non-experts are limited by their incomplete knowledge of the task domain. Therefore, they find it challenging to choose the appropriate criterion to focus on at a given stage in the task. For example, during the pilot stage of the Flud study, I noticed that the majority of the workers focused on distance-based criteria such as edge length and node distribution when I asked them to focus on the task without fixing a primary criterion-specific goal (or task mode). Focusing on low-priority
criteria like edge length while ignoring high-priority criteria like downward pointing paths led to unproductive work and, consequently, poor quality layouts. It indicates that novices may gravitate towards tasks that they perceive as easier instead of focusing on the advised task goals. This finding corroborates with prior work which shows that novices are inclined to choose a task based on personal preferences even if advised not to choose that task under a certain circumstances [88, 219].

To address this challenge, Flud computationally assigns task modes to the crowd worker at the start of the task. This strategy differs from the mixed-initiative collaboration in systems like Foldit where humans control the choice of task mode assignment. Furthermore, in an empirical study presented in Chapter 4, I found that assigning these task modes in the order of their priorities generates better results than assigning modes randomly. An implication of this finding is that crowdsourcing systems can computationally assign task modes to the crowd workers to achieve better results. This finding is especially important for human computation games where players often balance multiple goals and can choose from multiple task modes for a given task. Allowing the players to choose any task mode for a given scenario may lead to unfruitful work. Instead, to achieve better results, these systems can assign the task based on the mode selected by the player rather than allowing the player to choose any mode for a given task.

7.2.3 Implications for Graph Visualization Research

In my work presented in Chapter 4, I conducted a user study with seven participants with research backgrounds in computational biology to interpret the information flow in network visualizations. The results showed that layouts generated by methods that consider the downward pointing path criterion — i.e., Flud and Simulated Annealing— were clearly more
Chapter 7. Conclusions

effective and easy to use in comparison to other baseline layout methods. I also found that participants felt it was more strenuous to follow the signaling flow in the visualization if the paths were not downward pointing. Currently, the state-of-the-art methods use number of downward pointing edges as a quality metric for directed graphs. Based on my findings, I argue that merely counting the number of downward pointing edges [154] is not sufficient to capture the flow of information. Instead, my findings demonstrate that graph visualization methods should consider the proposed downward pointing paths criterion along with the other standard aesthetic criteria to produce effective layouts.

My dissertation empirically showed that Flud can generate effective visualizations for signaling pathways in cells. However, cell signaling pathways represent only one type of network data within biology where understanding information flow is critical. For example, identifying the path from an ancestral source node to target node in a phylogenetic network is important to understand the evolutionary history of a gene or species. Similarly, the information flow of ‘who eats whom’ in a pelagic food web network [212] is important to understand the mechanisms underlying the global degradation of coastal ecosystems. Disease progression pathways [78] are another type of networks where scientists need to understand information flow in order to reveal how a disease like cancer progresses, predict future behaviors, and inform targeted therapy. Other types of networks which may be effectively visualized using Flud include gene regulatory networks, protein co-complex networks, metabolic pathways, and nutrient signaling networks. Very recently, scientists have built a repository of SARS-CoV-2 virus-host protein interaction networks for network analysis to combat COVID-19 [146].

I believe the layout criteria that Flud embodies can also generalize beyond biology to other domains where visualizing information flow underlying the network is important. For example, supply chain diagrams provide a quick visual overview of the flow of merchandise from
7.3. Future Work

raw materials to manufacturers to consumers [64]. As another example, sociologists and epidemiologists study social networks and contact networks to understand how information or infectious diseases spread in a population [10, 62, 81]. Other types of networks where Flud can be used are communication networks [80, 101] and citation networks [71, 157]. Since Flud’s layout criteria only use network structure information and do not depend on any biological information, they may be used to lay out any type of directed network, though the per-criterion priorities may have to be re-adjusted depending on the domain requirements.

7.3 Future Work

While my work on Critter focused on creative tasks, some of the related findings may be generalizable to other applications as well. A promising application of Critter is creating effective checklists for ethical data management and modeling. Prior work in this area shows that just having a checklist of ethics guidelines is not enough, as it does not influence the practitioners’ decision [76, 135, 138]. However, my work on Critter suggests that using guidelines as rubrics for feedback can promote learning and better work in the longer term. Therefore, it would be interesting to explore how we can adapt the Critter system to influence the practitioners’ decision in the long term. The Critter system has been released as a part of an open source tool called Orchestra. Conducting field studies to investigate the challenges and potential use cases of Critter in other domains is a promising future direction of this work.

Another future direction of my dissertation work is to integrate the DeepLayout approach into the hybrid crowd-algorithm approach proposed in Chapter 4. However, this future work warrants a careful investigation. In the guidelines for human-AI interaction [5], Amershi et al. suggests that an AI agent should time its actions based on the user’s current task and
environment. However, in our graph layout task, it remains unclear when and how frequently the AI agent should intervene so that it does not remove the user’s ability to make novel contributions to the system’s goals. In my current work, Simulated Annealing and humans worked in alternate sessions asynchronously; a richer collaboration between AI and humans poses several open research questions. First, how can we design a human-AI collaborative experience that is enjoyable to the human? Can the human and AI collaborate synchronously in a single session? Second, if they collaborate synchronously, it remains unclear how can we elucidate a contribution made by a complex deep learning-based approach to a non-expert. Third, where do we draw a line between a support tool augmenting the user’s work and it hampering the user’s ability to make novel contributions to the system’s goals?

An opportunity for further improvement is to adapt Flud system to motivate volunteer participation. Given that Flud aims to aid scientific researchers, I expect an intrinsically motivated subset of volunteers would spend more time playing the game and make more improvements in a game session. Moreover, as the workers get familiar with the task, I expect high-performing workers to develop layout strategies they can reuse across different games (networks), as has been observed in other games like EteRNA [7] and FoldIt [33]. As a future work, it is worth exploring what enhancements are necessary to the Flud platform to motivate volunteer participation, and consequently, how volunteer worker performance compares with paid crowd workers from this study.

One idea for motivating volunteer participation is to add features to bolster Flud as a serious game. For example, prior work shows that a leaderboard can play an important role in games (e.g., Foldit [33], Peekaboom [215]) to motivate better performance [86]. However, in this work, Flud uses only a small number of game elements such as displaying the top score and clues and does not have a leaderboard. I decided not to use a leaderboard in my dissertation work since the participants were crowd workers on MTurk who are primarily motivated by
compensation rather than by competitive gameplay. Selecting, implementing, and testing appropriate game elements, such as a leaderboard, to motivate volunteer gameplay in a context where users are effectively collaborating with each other will be a promising future direction.
Bibliography


[29] Wei Chen, Fangzhou Guo, Dongming Han, Jacheng Pan, Xiaotao Nie, Jiazhi Xia, and Xiaolong Zhang. Structure-based suggestive exploration: a new approach for effec-


[63] Werner Funk, Vera Dammann, and Gerhild Donnevert. *Quality assurance in analytical


[84] Hammad Haleem, Yong Wang, Abishek Puri, Sahil Wadhwa, and Huamin Qu. Evalu-


[137] Toshiyuki Masui. Evolutionary learning of graph layout constraints from examples.


[208] Frank van Ham and Adam Perer. "Show Context, Expand on Demand": Supporting Large Graph Exploration with Degree-of-Interest. *IEEE Transactions on Visualization*


Appendices
Appendix A

Appendix for Chapter 4

A.1 Data Availability

The networks used in the experiments are available for download in Cytoscape.js JSON (.cyjs) format at https://doi.org/10.6084/m9.figshare.c.4970651.v1.

A.2 Computation of Per-criterion Scores

To prevent pathological layouts that move nodes indiscriminately far apart from each other, we set the per-criterion scores to zero if any node is moved outside a bounding box of fixed page width $w$ and height $h$. We chose $w = 5000$ and $h = 6000$ in our implementation. The aspect ratio of our bounding box approximately matches that of a page in a scientific journal, i.e., 0.8.

1. *Downward pointing paths:* We count the number of downward-pointing paths in a layout using a dynamic program that runs in time linear in the number of edges in the network. Note that this algorithm does not assign node positions in order to maximize the number of downward-pointing paths since this task will be undertaken by Flud players. The subgraph composed only of downward-pointing edges is acyclic. Hence, for any node $v$ in the network, we can compute $\pi(v)$, the number of downward-pointing
paths that start at \( v \) using the following recurrence:

\[
\pi(v) = \sum_{(v,u) \text{ is downward pointing}} \pi(u)
\]

Here, the sum is taken only over outgoing neighbors of \( v \) that have smaller \( y \)-coordinate than \( u \), i.e., if the edge \((v,u)\) is pointing downward. The base case is a node \( v \) that has no downward-pointing edges leaving it: \( \pi(v) = 0 \) in this case. We can compute the total number \( \pi(G) \) of downward-pointing paths in the graph by summing \( \pi(v) \) over all nodes \( v \) that have no downward-pointing edges entering them. Note the maximum possible value of \( \pi(G) \) is the number of paths (all directed paths, not just downward-pointing ones) in \( G \). We count this number \( \rho(G) \) using a depth-first search based algorithm. Although this algorithm has worst-case running time that is exponential in the graph size, it was very efficient in our experiments. Finally, we compute a normalized downward pointing score as follows:

\[
DP(G) = \frac{\pi(G)}{\rho(G)}.
\]

2. Non-crossing edge pairs: We count the number \( \chi(G) \) of edge pairs that do not cross by checking for each pair of edges whether they intersect or not. Since the maximum number of edge crossings possible in \( G \) is \( m(m-1)/2 \), we compute the non-crossing edge pairs score as

\[
EC(G) = \frac{2\chi(G)}{m(m-1)}.
\]

3. Edge length: A trivial but undesirable solution that satisfies this criterion is to place all nodes at the same location. Note that such a layout may have a high score despite the node distribution constraint (described next) because of the per-criterion priorities.
A.2. Computation of Per-criterion Scores

Therefore, we require every edge to have a minimum fixed length (300 pixels, in our implementation). We now define the cost \( c(e) \) of an edge to be equal to its length \( l(e) \) if \( l(e) \geq 300 \) or equal to a large number (say, 10,000), otherwise. We normalize the cost of each edge by the largest possible edge length (a diagonal of the screen) and compute the edge length score of a layout as

\[
EL(G) = \max \left\{ 0, 1 - \frac{1}{m} \sum_{e \in E} \frac{c(e)}{\sqrt{w^2 + h^2}} \right\}
\]

where \( w \) and \( h \) represent the width and height of the bounding box, respectively.

4. Node edge separation: To evaluate the fidelity of a layout to this aesthetic criterion, we compute the distance \( d(v, e) \) between every node \( v \) and every edge \( e \) in \( G \) as follows: we check if the projection of \( v \) into the line containing \( e \) falls within \( e \). If it does, \( d(v, e) \) is the distance between \( v \) and this projection. Otherwise, \( d(v, e) \) is the distance from \( v \) to the endpoint of \( e \) that is closer to \( v \). With these values in hand, we define the node edge separation score as

\[
NED(G) = \frac{1}{n} \sum_{u \in V} \min_{e=\{t,v\} \in E} \frac{d(u, e)}{\sqrt{w^2 + h^2}}
\]

where \( w \) and \( h \) represent the width and height of the bounding box, respectively.
A.3 Fine-tuning Procedure used to generate final Flud layouts

At the end of each sequence, we fine-tuned [87] the best layout so far by using it as the input to simulated annealing with a small value of the initial temperature \(T_0 = 10\). Due to the low initial temperature, the random moves are "local," i.e., a node can move only to nearby position. Additionally, we did not accept moves that decrease the score. We used the output of this step as to the final layout. We expected fine tuning to improve the distance-based components of the layout scores without dramatically affecting the components for downward pointing paths and edge crossings.

A.4 Computation of Bonus Compensation for the Crowd Workers

We assigned a budget to each criterion in proportion to its priority. This budget determined the amount of bonus a worker can earn while completing the task for the assigned criterion-specific mode. The bonus amount earned when a worker improves the score from \(s_1\) to \(s_2\), i.e., \(s_2 > s_1\), is

\[
(b + 1) \frac{\min\{s_2, s_{\text{target}}\}}{s_{\text{target}}} - (b + 1) \frac{\min\{s_1, s_{\text{target}}\}}{s_{\text{target}}},
\]

where \(b\) is the budget assigned to the given criterion and \(s_{\text{target}}\) is the target score we want the crowd workers to achieve. The exponential nature of the bonus computation ensured that the workers earn more money per point as they approach the target score. The goal is to motivate the workers to continue updating the layout, even though the task of improving the score gets harder as the worker gets closer to the target score. Ideally, we should aim to set...
\( s_{\text{target}} \) to an achievable value in order to ensure that the bonus is fair. For example, it is not possible for a crowd worker to achieve a score 10,000 (the maximum possible) if the network has many cycles. Therefore, we suggest using an achievable target (say, 2,000). Once the workers achieve the target score, the requester has the option to increase the budget and the desired score.

### A.5 Qualitative Comparisons of Layouts

We also qualitatively compared the best layouts generated using Flud and automated approaches (Figures A.1–A.6). In Spring-electrical model generated layouts (panel (d) in Figures A.1, A.3, and A.5), we observed that the nodes were well distributed in the layout space. However, the layouts had many upward pointing edges and was the hardest to interpret while looking for source-to-target paths. Dig-Cola, on the other hand, generated layouts with several downward pointing edges (panel (b) in Figures A.1, A.3, and A.5). But, it was still hard to read the layout because of many edge crossings and node-node and node-edge overlaps. IPSEP-Cola also generated layouts with several downward pointing edges but with lesser node-edge overlaps (panel (a) in Figures A.2, A.4, and A.6). However, the layouts still had many edge-crossings and the nodes were distributed such that it was hard to follow the source-to-target path in the layouts. In case of CerebralWeb (panel (b) in Figures A.2, A.4, and A.6), we noticed that it arranged the nodes in five layers in the following order: “cell surface”, “plasma membrane”, “cytoplasm”, “nucleus”, and “unknown”. While this order has a high-level correspondence with a receptor-to-TF path, CerebralWeb did not layout the nodes inside each layer in such a way that the final layout would contain many downward pointing paths. In contrast, the Flud and SA approaches (panel (a) and (c) in Figures A.1, A.3, and A.5, respectively) created layouts with many downward-pointing paths.
while achieving better separation among nodes and edges.

For instance, Figure A.3 shows the layouts generated for network G2, a network representing the Epithelial-to-mesenchymal transition (EMT) [192]. In the crowdsourced layout (Figure A.3a), we can clearly see the downward-pointing path from FGF to SOS/GRB2 on the right, whereas it is hard to clearly see this two-edge path in layouts generated using Dig-Cola (Figure A.3b), Spring Electrical (Figure A.3d), and CerebralWeb (Figure A.4b) method. We also note that Figure A.3b gives the false impression that paths from source nodes to the SNAI1 target node are very short. In contrast, it is clear from the crowdsourced layout (Figure A.3a) that SNAI1 is regulated by multiple upstream source nodes via various paths.
A.6 Supplementary Figures

Figure A.1: The highest-scoring layouts generated by Flud and baseline methods for network G1. These layouts are used to understand flow of signaling information from sources to targets in G1. Green and yellow colored nodes represent the source and target nodes, respectively. The upward pointing edges are shown in red whereas downward pointing edges are shown in blue color.
Figure A.2: The highest-scoring layouts generated by IPSEP-Cola and CerebralWeb baseline methods for network G1. These layouts are used to understand flow of signaling information from sources to targets in G1. Green and yellow colored nodes represent the source and target nodes, respectively. The upward pointing edges are shown in red whereas downward pointing edges are shown in blue color.
Appendix A. Appendix for Chapter 4

(a) Flud (Overall Score=429,439)

(b) Dig-Cola (Overall Score=39,349)

(c) SA (Overall Score=178,718)

(d) Spring Electrical Model (Overall Score=16,749)

Figure A.3: The highest-scoring layouts generated by Flud and baseline methods for network G2. These layouts are used to understand flow of signaling information from sources to targets in G2. Source nodes are green and target nodes are yellow. The upward pointing edges are shown in red whereas downward pointing edges are shown in blue color.
Figure A.4: The highest-scoring layouts generated by IPSEP-Cola and CerebralWeb baseline methods for network G2. These layouts are used to understand flow of signaling information from sources to targets in G2. Green and yellow colored nodes represent the source and target nodes, respectively. The upward pointing edges are shown in red whereas downward pointing edges are shown in blue color.
Appendix A. Appendix for Chapter 4

(a) Flud (Overall Score=1,196,882)
(b) Dig-Cola (Overall Score=35,295)
(c) SA (Overall Score=80,332)
(d) Spring Electrical Model (Overall Score=37,283)

Figure A.5: The highest-scoring layouts generated by Flud and baseline methods for network G3. These layouts are used to understand flow of signaling information from sources to targets in G3. Green and yellow colored nodes represent the source and target nodes, respectively. The upward pointing edges are shown in red whereas downward pointing edges are shown in blue color.
Figure A.6: The highest-scoring layouts generated by IPSEP-Cola and CerebralWeb baseline methods for network G3. These layouts are used to understand flow of signaling information from sources to targets in G3. Green and yellow colored nodes represent the source and target nodes, respectively. The upward pointing edges are shown in red whereas downward pointing edges are shown in blue color.
Figure A.7: Distributions of EC, EL, ND, and NED per-criterion scores of layouts created by different approaches.
Figure A.8: Plots of the time taken by three game sequences of different approaches to increase the total layout score. Each row corresponds to a network. The $x$-axis represents the time taken in hours to reach a particular score and the $y$-axis represents the score. The circle and square markers correspond to crowd workers and simulated annealing respectively.
Figure A.9: Plots illustrating the impact of criterion-specific modes and clues. A) Distribution of number of times crowd workers moved nodes with or without a clue in a criterion specific mode. B) Distribution of average improvement in per-criterion score per move achieved by crowd workers with or without using the criterion-specific clue.