University of London
Imperial College of Science, Technology and Medicine
Department of Electrical and Electronic Engineering

Advances in network visualisation with an application to serious games

Jonathan Xiang-Sheng Zheng

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy of the University of London and
the Diploma of Imperial College, August 2020

Abstract

This thesis concerns the visualisation of networks, through an in-depth study into the node-link diagram representation. Three subtopics are explored within this space. The first is the problem of node layout, where the optimisation of a popular energy function, known as stress, is improved through an algorithm known as stochastic gradient descent. The second is the method of edge bundling, where the idea of hierarchical edge bundling is explored in the absence of a known ground truth hierarchy. Its similarity to a topologically lossless bundling method known as power-confluent drawing is then leveraged, in order to improve technical problems with the underlying algorithms. The final topic is an engineering application in the form of a serious game called EcoBuilder, which utilises the node-link diagram to visualise the dynamical behaviour of food webs. Its purpose is to crowdsource research through a citizen science approach, with outcomes in both visualisation and mathematical ecology.

Acknowledgements

This entire project started back in 2015, as a short proposal on a long list of Masters projects, titled “Ecosystems visualisation and game for scientific outreach”. It immediately jumped out at me as my definite first choice, and I was more than happy when I found that it had been allocated to me. I knew at the time that the project would be fun and interesting, but I certainly did not expect that, almost five years later, I would be writing up the journey as a PhD thesis. I would therefore like to first thank my supervisors, Dan and Samraat, for sticking with me all this way. It has been a pleasure to work with both of you, and I cannot imagine having anybody else guiding me through this path.

I am also grateful to my viva examiners, Daniel and James, for providing excellent and necessary feedback over my five hour viva, which I am sure was as exhausting for you as it was for me.

I would also like to thank all of the gang in the lab, without whom it would have been orders of magnitude more difficult to stay positive through the tough times. Thank you Pamela for expanding and fixing my world view, Juil for always making us laugh, Daphne for your singing and positivity (and memes), Razvan for your kindness and cakes, Sara for your caring nature and comedic genius, Lotte for sacrificing your career as a pop star to bring your positivity to us instead, Jean for being okay at chess but great at weddings, and of course Yang for dealing with me as a housemate for the past few years, even through what has been an arduous 2020 to say the least. I wish every one of you the very best for the coming years and hope to see you all again.

I must also thank Mum for always being there for me, Dad for inspiring my scientific side, and Jennifer for being an exceptional partner in crime. A warm thank you also goes to Ka Yan, for supporting me and keeping me afloat when work was tough, and when I had lost sight of the light at the end of the tunnel. You mean the world to me.

A huge thank you also goes to fellow PhD student Hsi-Cheng for helping me with the ecological side of EcoBuilder. The game would not look anything like it does now without your help, including the entire genre of the game being puzzle instead of action! The final release would also not have been possible without help from the members of Neural Reckoning and Pawar Lab, who helped to playtest the game, and volunteered to set up stands at various science festivals to collect player feedback.

Finally, I would also like to thank all the players of EcoBuilder who made the work in the final chapter possible, and for surpassing all expectations I had for how high you would score. It has been a wonderful experience to lead an initiative for crowdsourcing research, and I am proud to have been able to make little citizen scientists out of you all. My only hope is that you all had fun and learned something along the way, just as I did on my journey towards completing this thesis.

"What bothered me was, I thought he must have *done* the equation. I only realized later that a man like Wheeler could immediately *see* all that stuff when you give him the problem. I had to calculate, but he could see."

Richard P. Feynman

Contents

Abstract i

Acknowledgements iii

1 Introduction 1

1.1 Thesis roadmap 5

1.2 Statement of Originality 6

1.3 Copyright Declaration 6

1.4 Publications 7

2 Stress minimisation 8

2.1 Background 11

2.1.1 ~~Force-directed~~ Optimisation algorithms 14

2.1.2 Multidimensional scaling 21

2.1.3 Majorization 22

2.1.4 Constraint relaxation 24

2.1.5 Large graphs 26

2.2 Stochastic gradient descent 28
2.2.1 Step size annealing 32
2.2.2 Randomisation 38
2.2.3 Systematic study 40
2.2.4 Smart initialisation 46
2.3 Sparse approximation 47
2.4 Applications 51
2.4.1 Radial layout 51
2.4.2 General multidimensional scaling 53
2.5 Discussion 55

3 Hierarchical edge bundling **58**

3.1 Background 58
3.1.1 Hierarchical bundling 61
3.1.2 Clustering 63
3.1.3 Hierarchical clustering 67
3.1.4 Random walks as dissimilarities 72
3.2 Experimental study 77
3.2.1 Results 79
3.2.2 Pruning 82
3.3 Discussion 86

4 Power-confluent drawing **89**

4.1 Background 90

4.1.1 Power graph decomposition 91

4.1.2 Confluent drawing 93

4.2 Improvements 94

4.2.1 Greedy heuristics 95

4.2.2 Node splitting 98

4.2.3 Short-circuits 100

4.2.4 Hierarchical routing 102

4.2.5 B-splines 105

4.2.6 Directed graphs 108

4.2.7 Planarity 109

4.3 Discussion 112

5 EcoBuilder **114**

5.1 Background 114

5.1.1 Predator–prey interactions 116

5.1.2 Metabolic scaling 120

5.1.3 Food web topology 125

5.1.4 Dynamic graph layout 130

5.1.5 Citizen science 133

5.2 Experimental design 136

5.2.1 Learning World 137
5.2.2 Research World 138
5.2.3 Objectives 139
5.3 Results and analysis 141
5.3.1 Objective 1 142
5.3.2 Objective 2 145
5.3.3 Discussion 150
5.4 Appendix: User interface 153
5.4.1 Food web visualisation 155
5.4.2 Heads-up display 160
5.4.3 Data collection 166

Conclusion **168**

Bibliography **170**

List of Tables

3.1 Benchmark graphs used for the experiment in Section 3.2 78

4.1 Experimental results on greedy power graph decomposition 97

5.1 Values of constants used for the model in Section 5.1.1 124

List of Figures

2.1 A gallery of graph representations 9

2.2 An example output of a planarity-based algorithm 13

2.3 A gallery of node layout methods 15

2.4 Illustration of the distance constraint in Equation 2.13 25

2.5 Results against majorization for `1138_bus` and `dwt_1005` 31

2.6 Pseudocode for stochastic gradient descent 31

2.7 A comparison of different annealing schedules 34

2.8 A comparison of annealing schedule parameterisations 34

2.9 A comparison of methods for term order randomisation 39

2.10 A systematic comparison against majorization over 243 graphs . . 41

2.11 A summary of the results in Figure 2.10 42

2.12 The one graph in the test set where majorization performs better . 43

2.13 Stress measured on a further selection of graphs 45

2.14 Pseudocode for sparse stochastic gradient descent 48

2.15 Results against majorization using sparse stress 50

2.16 A gallery of large graphs layed out using sparse stress 50

2.17 The London tube map with a focus on Green Park station 52

2.18 A graph `lesmis` with node colours embedded in RGB space 54

3.1 The Unity game engine, visualised using the Unity game engine . . 59

3.2 Two examples for why clustering is difficult to define 65

3.3 A hierarchical edge bundling pipeline 68

3.4 Experimental clustering results for `karate` and `footballTSE` . . . 80

3.5 Experimental clustering results for `caltech` and `lfr` 81

3.6 Hierarchical edge bundles without pruning 83

3.7 Final hierarchical edge bundled diagrams 85

4.1 A power-confluent drawing pipeline 90

4.2 Pseudocode for improved power graph decomposition 98

4.3 Examples of the node split problem 99

4.4 Examples of the short-circuit problem 100

4.5 A full version of Figure 4.4 101

4.6 A fixed method for confluent drawing 103

4.7 Pseudocode for power-confluent drawing 104

4.8 B-spline basis functions 107

4.9 A strict confluent drawing that is not power-confluent 111

4.10 Examples of networks drawn with the power-confluent algorithm . 113

5.1 EcoBuilder displayed on two mobile devices. 115
5.2 A plot of metabolism against individual body mass 121
5.3 An illustration of food web dimensionality 128
5.4 Learning World and Research World 137
5.5 A side-by-side comparison of trophic level constraints 140
5.6 Bar charts showing player progressed 142
5.7 Histograms showing the time spent on each level 144
5.8 Results from Research World Level 1 146
5.9 Results from Research World Level 2 147
5.10 Results from Research World Level 3 148
5.11 Player demographics 151
5.12 A labelled screenshot of EcoBuilder gameplay 154
5.13 Player registration and data collection 166

Chapter 1

Introduction

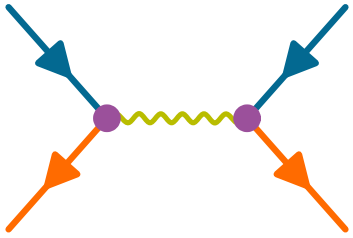
In his 1985 book, *"Surely you're joking, Mr. Feynman!"*, the physicist Richard Feynman writes about an experience he had as a graduate student at Princeton University. One day, Feynman was scheduled to give a talk about his work at an ordinary weekly seminar, but the stars aligned to bring a host of famous names together into the audience for that week, including John von Neumann, Wolfgang Pauli, and Albert Einstein. The story itself focuses on Feynman's understandable surprise and nervousness about the situation, but he also noted an interesting detail about these great minds – they shared an uncanny ability to predict the outcomes of complex mathematical processes, without having to resort to using algebra to work through the equations by hand.

Feynman refers to this ability as being able to *see* the physics, and was no doubt part of the inspiration behind *Feynman diagrams*, an alternative representation of the complex equations that describe interactions between subatomic particles.

For example, the equation

$$\mathcal{M} = (O_1 ie\gamma^\mu I_1)\left(\frac{-ig_{\mu\nu}}{p^2}\right)(O_2 ie\gamma^\nu I_2)$$

and the Feynmann diagram



are equivalent, with both similarly colour-coded to match the corresponding elements of the interaction they are describing. Both the equation and the diagram can be classified as a *representations* of the abstract concept of a mathematical model, but it should be much clearer in the diagram that what is being described is the movement of two particles as they exchange a photon.

The subject of this thesis is not physics, so the details of the above interaction are not important. What is important is the fact that variations of these diagrams, with slightly different rotations and symbols, can closely approximate the behaviour of any subatomic particle interaction, whilst remaining simple enough to make complex and unwieldy integral expressions digestible. They helped open a door to understanding the natural world, revolutionising the field of quantum electrodynamics upon their introduction (Kaiser, 2005). Feynman's contributions to science were eventually recognised by a Nobel prize shared with Shinichiro Tomonaga and Julian Schwinger, and the calculations behind their work could not have been possible without these diagrams (Kaiser, 2009).

Why does this matter? Because the modern world has brought with it a powerful tool with the potential to further unlock this ability to *see;* not everybody can be

a genius like Einstein or Feynman himself, but nowadays almost everybody does have access to something they did not: a personal computer. Tables of data no longer need to be plotted by hand, because a computer can do it automatically, without error, and on volumes of data impossible for any person to handle in multiple lifetimes. But presenting this information in the best format to support human understanding is not trivial, because despite the brain's amazing ability for pattern recognition, it is also easily overloaded and often misled. The study of what representations work best for humans, *visualisation*, is the real subject of this thesis.

More specifically, the focus will centre upon on the visualisation of *networks*. Any set of relationships between entities can be classified as a network. Commonly studied examples include networks of relationships on social media, biological metabolic pathways, and citation patterns between research publications. They can be found everywhere and are becoming an incredibly important data structure, but it is far from a trivial task to faithfully represent their underlying patterns, especially when the size of the network approaches the order of thousands or even millions of nodes. The majority of this thesis concerns the algorithms behind modern methods of network visualisation.

Another benefit of computers has been the development of user interfaces, which have flourished as a result of the massive recent growth in consumer electronics. This has brought a whole new language to the world, where people have learned the ability to communicate with machines, now small enough to carry in their pockets. There is a whole new generation of society who have grown up with these devices to become 'computer-literate', granting them access to the world of *interactive visualisation*, which dynamically responds to the user depending on what sections they wish to view, and how they wish to view them. Despite this, static diagrams are still invaluable, not least because they can be printed on paper, which

is still the most accessible form of media in the world. A well-organized static visualisation also allows the viewer to digest the information in their own time, and in the logical order that suits them best. This is just as a painting composed by a talented artist, with the right details spread across the right locations, can be observed and appreciated for just as long as a good movie. As such, the majority of the work done here focuses on static visualisation. However it would be wasteful, if not foolish to ignore the wide realm of possibilities that interactive media can offer.

The final chapter of this thesis therefore explores an application of the algorithms developed in preceding chapters, in the form of a research-oriented video game called *EcoBuilder*. The goal of the player in the context of EcoBuilder is to construct network of species connected by predator-prey interactions, whilst preventing any species from going extinct. The result of which species go extinct and which survive is determined by a dynamical system of equations used to model real-world ecosystems, and so the research outcome of the game is to crowdsource research into these dynamical systems through an approach known as *citizen science*.

EcoBuilder is the thread that pulls together the entire narrative of this body of work. Despite being the subject of the *final* chapter, the goal of producing a video game was the *initial* target set for the PhD behind this thesis. At their core, video games are simply interactive visualisations, with the one primary difference that they present the user with an objective to complete. Within every game there is an underlying virtual world for the player to understand and interact with; the structure of this virtual world within EcoBuilder is shaped in the form of networks, and so by definition a visualisation of such networks was required to reach the target of producing the game.

However, the ensuing rabbit hole opened by the topic of network visualisation re-

sulted in the bulk of the work here to be done parallel to the overarching goal of a video game. The task of building an accurate depiction of any network, let alone one with underlying dynamics like an ecosystem, is an intricate undertaking, and in hindsight it is no surprise that its exploration led to interesting and fruitful avenues of research. Not all of these avenues eventually found themselves included within EcoBuilder itself, but hopefully the contributions will nevertheless be of use to the visualisation community at large.

1.1 Thesis roadmap

This thesis is organised as follows. There are four main chapters, each of which explores a topic in the world of network visualisation. Chapter 2 concerns the layout of nodes in the node-link representation of a network. It presents the application of an algorithm known as stochastic gradient descent to the problem of node layout, which outperforms the previous state-of-the-art for a popular objective function known as stress. Chapters 3 and 4 concern the presentation of the links in a node-link diagram, specifically the idea of curving links to reduce clutter. Both chapters present methods that leverage the use of an auxiliary graph to inform this curvature. The difference between the two chapters is the method for constructing this auxiliary graph: the first explored in Chapter 3 uses hierarchical clustering, and the second in Chapter 4 uses power graph decomposition. Chapter 5 presents an application of network visualisation algorithms to a research-oriented video game called EcoBuilder. The objectives aimed at from the design of the game are first presented, and the research outcomes from releasing it to the public are then reported.

The broader literature review for the topics studied is split up across these chapters, in Sections 2.1, 3.1, 4.1, and 5.1, each titled Background. A general context for

the fields studied in this thesis as a whole can largely be garnered from reading each of the three individual chapters together, if desired.

1.2 Statement of Originality

This is to certify that, to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any other degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

1.3 Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution Non-Commercial No Derivatives license (CC BY-NC-ND).

Under this licence, you may copy and redistribute the material in any medium or format on the condition that; you credit the author, do not use it for commercial purposes and do not distribute modified versions of the work.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

1.4 Publications

The work in Chapter 2.2 was previously published in Zheng et al. (Zheng, Pawar, and Goodman, 2019c). The layout algorithm developed there was also applied to food webs in Ho et al. (Ho et al., 2019). An early version of the work presented in Section 3.1.1 was previously published in Zheng et al. (Zheng, Pawar, and Goodman, 2018) The work presented in Section 4.1.1 has previously been published in Zheng et al. (Zheng, Pawar, and Goodman, 2019b). An early version of the game presented in Chapter 5 was published in Zheng et al. (Zheng, Pawar, and Goodman, 2019a)

Chapter 2

Stress minimisation

If one were to ask a random group of people to draw a network/graph on a piece of paper, it is likely that most would draw dots to represent the nodes, and lines joining the dots to represent the edges. This is a representation so intuitive that it is often synonymous with the abstract concept of a graph entirely. This is the reason why this 'join-the-dots' representation, known as the node-link diagram, is the most commonly studied and applied (Ghoniem, Fekete, and Castagliola, 2004), and is also why it has been chosen for the purposes of this thesis.

The contribution of this first chapter is entirely focused on algorithms for positioning the nodes in a node-link representation, specifically through the numerical optimisation of a popular objective function known as *stress* (Zheng, Pawar, and Goodman, 2019c). However, it is useful to have an idea of the other possible graph representations in order to gain a broader view of what network visualisation can be. A classical example of this includes the *matrix plot*, which is a grid where each vertex is represented by a row and a column, and each edge is a dot filled in at the intersection of a row and column (Liiv, 2010). Specialised types of networks can also have similarly specialised representations. Trees, for example, can be depicted as packed rectangles (Johnson and Shneiderman, 1991) or circles

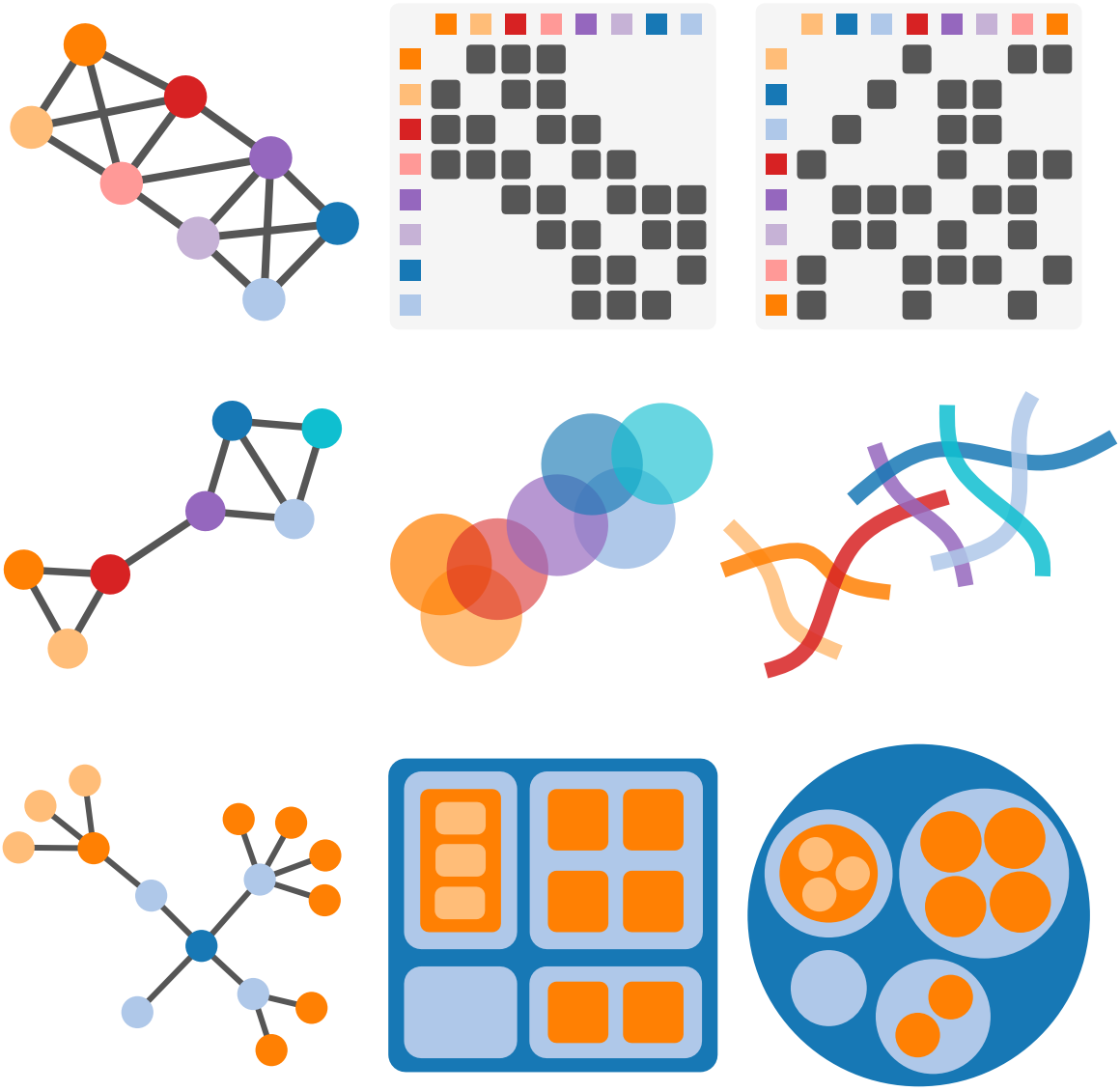


Figure 2.1: A gallery of different graph representations, where left column contains node-link diagrams for three different graphs, and each row shows the same graph but in different representations. The top row shows matrix plots with rows ordered nicely and shuffled; the middle shows disc and string intersection graphs; the bottom shows rectangular and circle packed treemaps.

(Wang, Wang, et al., 2006), in what are known as *treemaps*. Graphs with low *boxicity*, such as food webs (Eklöf et al., 2013), can be drawn as *intersection graphs* of overlapping lines or rectangles or cuboids.[1] A related and more common example is the *disc graph*, where nodes are represented by circles and edges exist if the circles overlap. This sees widespread use as a Venn diagram, but usually without the connotation of network structure. A more obscure example is *string graphs*, where each vertex is represented by a (possibly curved) line, and edges exist between vertices whose lines intersect.

A gallery of such examples can be seen in Figure 2.1. Each of these each of these representations has its unique benefits and downsides: matrix plots can show a very dense amount of data in a small space, but are very dependent on the ordering of rows and columns (Liiv, 2010), as it can be seen in Figure 2.1 that shuffling the order of rows and column can completely hide the underlying structure. A survey of algorithms that can be used to find insightful orderings can be found in Behrisch et al. (2016). The intersection-style graphs are intuitive and do not require edges to be explicitly drawn, thus saving on visual clutter. However they cannot be drawn for all graphs for either the circle (McDiarmid and Müller, 2014) or string (Schaefer, Sedgwick, and Štefankovič, 2003) representations. Treemaps effectively convey the idea of a hierarchy, where the root node completely envelops each of its children in a recursive manner. On their own they can only visualise trees, but have been effectively used in conjuction with clustering algorithms to apply them to general graphs; examples of this include *graph thumbnails* (Yoghourdjian et al., 2018), or *power graph decomposition* (Dwyer et al., 2014) which will be studied later in Chapter 4, Section 4.1.1.

Even within the subfield of node-link diagrams, there is a wide variety of options available. Examples include *arc* or *chord diagrams*, where nodes are placed on a

[1] Or hypercuboids, although the usefulness of that for visualisation is likely limited.

line or around a circle, respectively. Links are then added by drawing the eponymous arcs or chords between nodes. A method that has recently gained popularity is the *hive plot*, a simple but effective variant of parallel coordinate plots (Krzywinski et al., 2012), which places nodes on radial lines to draw curves between them, in a similar fashion to radar charts (Porter and Niksiar, 2018). An important subtlety is that each node may or may not be placed on more than one line, and the order in which the nodes are spread across the line is also a conscious choice. This customisability is where the power of such a method lies.

Hopefully the above examples give a taste of how varied the representation of a graph can be, even if the work in this thesis focuses its attention solely on the node-link diagram. The remainder of this chapter will specifically concern the *layout* of nodes in such a visualisation. And while there will be a focus on automatic node layout, it is worth noting that how humans tackle the problem has also been studied empirically (Purchase, Pilcher, and Plimmer, 2010; Purchase, 2014).

2.1 Background

The formal study of node-link diagrams dates back to least the 1920s. For example, Fáry's theorem is a famous proof that any planar graph, defined as a graph that can be drawn without any intersecting links, can always be drawn in a planar way without needing links to be curved. This proof is attributed to Fáry (1948), and was independently discovered by a host of other authors in the same era (Steinitz, 1922; Wagner, 1936; Koebe, 1936; Stein, 1951). The ensuing development of actual *algorithms* for network layout emerged around the 1960s, with its seminal work widely attributed to the barycentre algorithm of Tutte (1963).

Tutte's algorithm is very simple, and boils down to solving a system of linear equa-

tions, each of which strives to set a single node to the barycentre (i.e. mean position) of its neighbours. This is defined as

$$\mathbf{X}_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} \mathbf{X}_j \tag{2.1}$$

where $\mathbf{X}_i$ is the i^{th} row of the $|V| \times k$ matrix $\mathbf{X}$, with V being the set of vertices in the graph and k the dimensionality of the layout (usually two); $N(i)$ is the set of vertices neighbouring i. Throughout this chapter, $\mathbf{X}$ will represent the coordinates of nodes in a layout. Note that a necessary initial step is to fix the position of a selection of nodes around the boundary of the drawing, in order to avoid the trivial solution of placing all nodes in the same position. The powerful insight that Tutte revealed with his algorithm was a remarkable mathematical theorem attached to it. He proved that this barycentre algorithm will always produce a planar drawing, in the specific case that the graph is planar and tri-connected (i.e. the graph cannot be disconnected by removing two vertices, no matter which two are removed).

This algorithm has served as the springboard for two main branches of network layout algorithms. Since planarity has been shown to be one of, if not the most important markers for readability in node-link diagrams (Purchase, 1997), the first branch is a line of planarity-based algorithms which can guarantee planar node layouts. Examples include the algorithm of Read (1987) which recursively removes nodes and adds dummy edges to maintain a triangulated graph at each step. A problem with this method and Tutte's is *node resolution*, which means that nodes may be placed exponentially close to each other, rendering the graph impossible to read in certain areas.

A breakthrough for this problem came in 1990 by de Fraysseix, Pach, and Pollack (1990) who described the first layout algorithm to achieve an asymptotically optimal node resolution of an $\mathcal{O}(|V| \times |V|)$ grid, and in linear time (Chrobak and Payne,

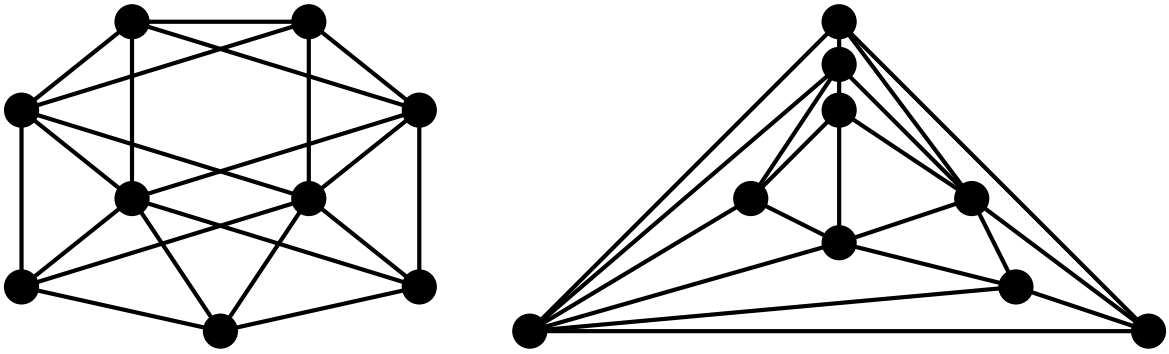


Figure 2.2: A graph with nodes positioned symmetrically (left) and without crossings (right) from using the algorithm of de Fraysseix, Pach, and Pollack (1990). Despite the planarity of the output, the poor angular resolution makes the drawing more difficult to read.

1995). An example output of this algorithm can be seen in Figure 2.2, which works by constructing a *canonical ordering* of the graph (Zhang and He, 2005) to order nodes on the x-axis, and from there y-axis positions can be chosen such to avoid crossings.

Improvements and refinements have been made with this algorithm in mind (Zhang and He, 2005), but such methods suffer from poor *angular resolution*, which means that adjacent edges form small angles between each other, another property that has been shown to negatively impact the readability of layouts (Purchase, 1997). Such an example of poor angular resolution is illustrated in Figure 2.2. This issue exists until this day (Eades and Hong, 2012), and is a primary reason why most network visualisation software uses algorithms based on a second, less mathematically rigorous branch of algorithms also inspired by Tutte.

This second branch comes from an intuitive interpretation of Tutte's algorithm: that each edge is analogous to a spring of zero natural length, where the solution to Tutte's system of linear equations can be seen as the point at which the elastic energy of these springs, according to Hooke's law (Hooke, 1678), is minimised.

This energy is defined as

$$\text{energy}(\mathbf{X}) = \sum_{\{i,j\} \in E} ||\mathbf{X}_j - \mathbf{X}_i||^2 \tag{2.2}$$

where E is the set of all edges. Differentiating with respect to the position of a single node $\mathbf{X}_i$ results in

$$\frac{d}{d\mathbf{X}_i}\text{energy}(\mathbf{X}) = \sum_{j \in N(i)} -2(\mathbf{X}_j - \mathbf{X}_i) \tag{2.3}$$

where it can be seen that setting the left-hand side to zero results in Equation (2.1), corresponding to an embedding of minimum global energy in the system.

This interpretation has been taken and advanced to alleviate the resolution problem present in planarity based methods, by introducing the trade-off of foregoing mathematical rigour. This is done by using human intuition to formulate variations on Equation (2.2), in what are known as *force-directed* algorithms.

2.1.1 ~~Force-directed~~ Optimisation algorithms

This section will present an overview of the various methods that have sprouted from this second branch of algorithms, around which the work in this chapter is based. All such algorithms will also be framed within the wider context of *optimisation*, a framework which will tie together otherwise loosely-connected threads in a logical taxonomy. It will also lead more cogently into the novel contributions described in Section 2.2. A gallery of the algorithms to be described here is presented in Figure 2.3.

This interpretation of framing the layout problem as optimisation is far from new, for example a technique known as simulated annealing has been applied to graph

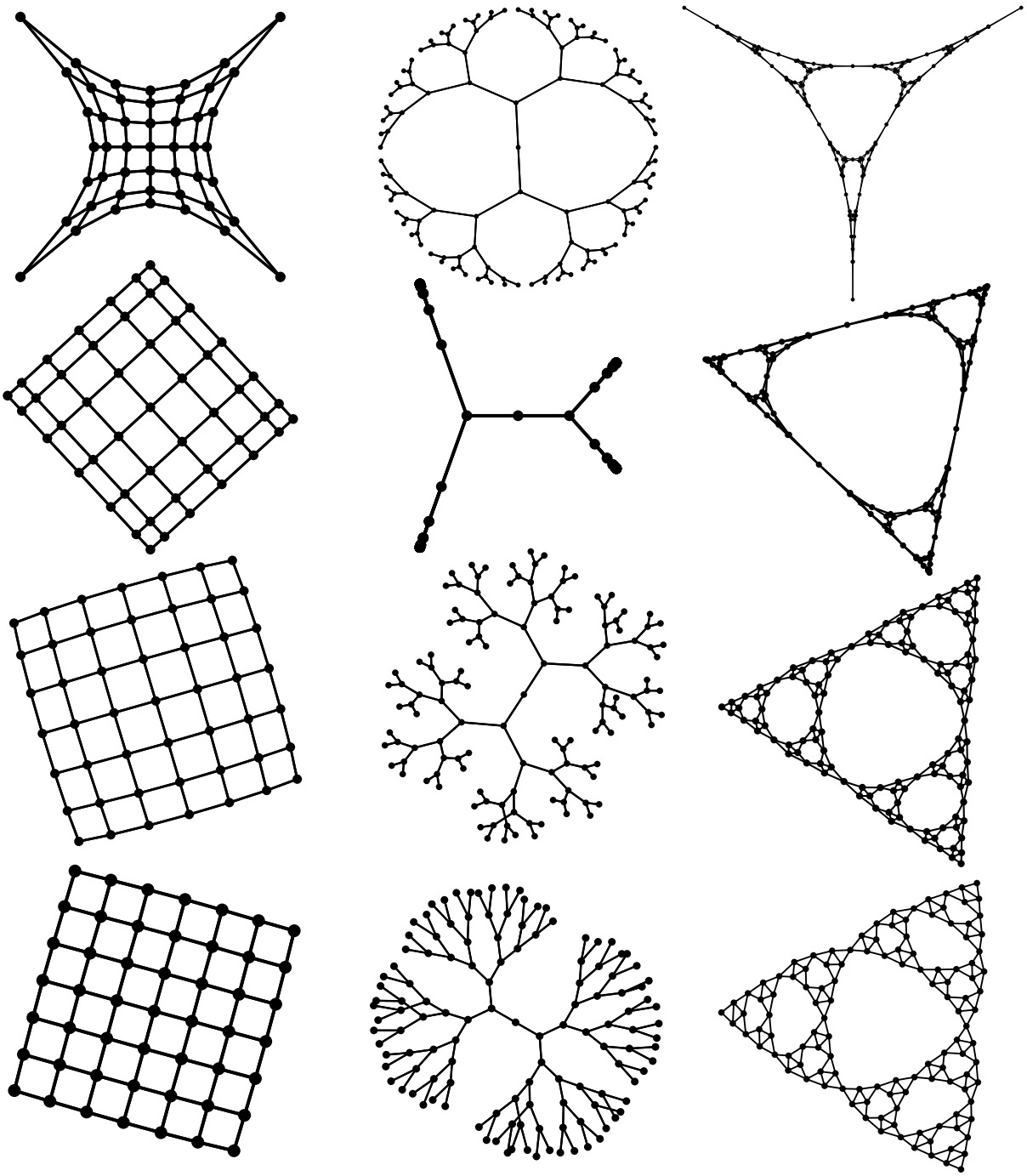


Figure 2.3: Three graphs visualised with four different force-directed algorithms from Section 2.1.1. Graphs are, from left to right, a grid of width seven, a binary tree of depth seven, and a Sierpiński triangle (Sierpiński, 1915) with five sub-triangle recursions. The algorithms used are, from top to bottom, Tutte (2.1), spectral (2.6), Eades (2.4), and stress (2.8). The nodes pinned to the edge of the layout for Tutte are the four corners of the grid, all leaf nodes in the binary tree, and the three corners of the biggest Sierpiński triangle.

drawing since the 90s (Davidson and Harel, 1996) which is strongly rooted within the field of optimisation. However it is important to acknowledge the word 'force' as a slight misnomer, as physical simulations are not used in their implementation, a fact also emphasised by Fruchterman and Reingold (1991) in their description of their popular layout method. 'Force-inspired' may have made for better terminology.

The earliest of these algorithms was developed by Eades (1984) who, inspired by techniques for positioning transistors on integrated circuits (Quinn and Breuer, 1979), made two modifications. The first was to alter the edge springs in order to give them a non-zero natural length, avoiding having to fix an often arbitrary selection of nodes around the boundary. The second was to introduce a repulsive force between pairs of non-adjacent vertices to spread nodes evenly around the drawing. The combination of these forces on a single node i is defined as

$$\frac{d}{d\mathbf{X}_i}\text{energy}(\mathbf{X}) = -\sum_{j \in N(i)} c_1 \log(||\mathbf{X}_j - \mathbf{X}_i||)\overrightarrow{\mathbf{X}_{ij}} + \sum_{j \notin N(i)} \frac{c_2}{||\mathbf{X}_j - \mathbf{X}_i||^2}\overrightarrow{\mathbf{X}_{ij}} \quad (2.4)$$

where $\overrightarrow{\mathbf{X}_{ij}} = \frac{\mathbf{X}_j - \mathbf{X}_i}{||\mathbf{X}_j - \mathbf{X}_i||}$, i.e. the normalised vector pointing from i to j, and c_1 and c_2 are constant parameters determining the relative strengths of the forces. The first summation defines the 'springs', where the logarithm attempts to maintain the spring at unit length by flipping to negative if the node pair gets too close together.[2] The second summation is the repulsive force, which always pushes i away from j if there does not exist an edge between them, and decays according to an inverse square function analogous to charged electrons obeying Coulomb's law (de Coulomb, 1785).

There are two important aspects to notice here. The first is that the left-hand side

[2] Hooke's law was abandoned by Eades because *"Experience shows that Hookes Law (linear) springs are too strong when the vertices are far apart; the logarithmic force solves this problem"* (Eades, 1984).

is not the energy itself, but its derivative. The second is that this derivative can no longer be straightforwardly solved as in Equation (2.1) because it has become *non-linear*. How then is energy minimised? Through a method known as *gradient descent*, which is as simple as iteratively moving nodes in the direction opposite to the derivative in Equation (2.4) according to

$$\mathbf{X}_i \leftarrow -\eta \frac{d}{d\mathbf{X}_i} \text{energy}(\mathbf{X}) \tag{2.5}$$

where η is a constant parameter. This operation, despite its simplicity, is theoretically proven to find a minimal energy embedding (Cauchy, 1847), although it is important to note that this embedding may not be globally optimal because the energy function defined by (2.4) is *non-convex* and therefore may contain many *local minima*. Many of the concepts introduced in this paragraph will be elaborated upon in Section 2.1.2.

This optimisation through gradient descent interpretation is not how this type of algorithm is commonly presented, as force-directed algorithms are often categorised into two families: force-balancing and energy-minimising models (Ortmann, Klimenta, and Brandes, 2017; Brandes, 2001b). Eades' model fits into the former, while others fit into the latter by directly defining an energy function such as in (2.2). Energy is then minimised by gradient descent as in Equation (2.5) or by another optimisation technique, as will be further elaborated upon in Section 2.1.2. Consequently, with the view that the action of 'force-balancing' is equivalent to gradient descent on another energy function, it is made clear that these two families are equivalent as they both strive for the same goal: minimising energy.

This optimisation-centric viewpoint also allows a variety of linear algebra-based methods to be grouped into the same taxonomy. The algorithm of Harel and Koren (2004) first embeds a graph in a high-dimensional space, determined by a co-

variance matrix of graph-theoretic or shortest-path distances, and then finds an optimal projection down to two dimensions by finding the eigenvectors of this this matrix. This is commonly known as principal components analysis (PCA) (Pearson, 1901). The *classical scaling* approach of Brandes and Pich (2007) is based on constructing the high-dimensional embedding from a double centered *distance matrix* between high-dimensional points and then performing PCA. The intuition behind this method is that it optimises the inner product between low- and high-dimensional distances.

The *spectral* approach, originally developed by Hall (1970) and largely ignored by the field until Koren (2003), uses a graph Laplacian as the high-dimensional matrix instead of shortest-paths, upon which PCA is again applied. This approach is particularly closely related to Tutte's algorithm, which in fact utilises the same graph Laplacian when solving for Equation (2.1). The eigenvalue approach even optimises the same energy function, but requires the variance of the embedding to be non-zero, thereby avoiding the need to fix certain nodes around the boundary (Koren, 2003). In mathematical terms, its energy upon a single dimension of the layout in the column vector $\mathbf{x}$ can be derived as

$$\begin{aligned} \text{energy}(\mathbf{x}) &= \sum_{\{i,j\}\in E} w_{ij}(\mathbf{x}_j - \mathbf{x}_i)^2 \\ \text{var}(\mathbf{x}) &= \frac{1}{n}\sum_i (\mathbf{x}_i - \bar{\mathbf{x}})^2 = 1 \end{aligned} \tag{2.6}$$

where w_{ij} is an optional weight given to each edge (previously all set to 1 in (2.1)) and $\bar{\mathbf{x}}$ is the mean of $\mathbf{x}$. The one-dimensional layout that minimises energy is equal to the second smallest eigenvector of the Laplacian matrix $\mathbf{L}$, defined such that

$$\mathbf{x}^T\mathbf{L}\mathbf{x} = \text{energy}(\mathbf{x}) \tag{2.7}$$

where $\mathbf{x}^T$ denotes the transpose of $\mathbf{x}$. Subsequent dimensions are found by cal-

culating subsequent eigenvectors of $\mathbf{L}$, often through power iteration (Koren, 2003). All of these methods boil down to finding the eigendecomposition of a certain matrix which describes the graph in a different way, and each set of eigenvectors describes an *optimal projection* into a lower dimension.

The linear nature of such methods is a double-edged sword, as it allows them to be calculated and/or approximated quickly and precisely, but they all suffer from the same issue that resulting layouts under-represent local detail (Brandes and Pich, 2008), especially in examples such as the binary tree in Figure 2.3, second row. However their effectiveness at capturing global structure consistently has led to them being commonly used as an initialisation step (Brandes and Pich, 2008; Dwyer, 2009). This will become important later when discussing the performance of a novel optimisation process in Section 2.2.4.

Circling back to non-linear methods, there have been many attempts at improving Eades' original formulation. A widely-used alternative, developed in 1991, is the method of Fruchterman and Reingold (1991) who revert springs back to having a natural length of zero, but balance this out by applying the repulsive force between all pairs of vertices. Counter-intuitively, increasing the number of repulsive terms also makes the summation more efficient to calculate, by taking inspiration from techniques used in physical simulations (Hachul and Junger, 2004; Hu, 2005). More detail on scaling layout algorithms to large graphs can be found in Section 2.1.5. Another notable attempt came from Frick, Ludwig, and Mehldau (1995), who altered the calculations to never require a square root operation, and introduced a number of extra heuristics to speed up the convergence of the optimisation procedure in Equation (2.5). These heuristics were introduced on an intuitive and ad hoc basis, but have been shown to greatly speed up convergence in practice (Brandes, 2001b).

Augmentations to force models with the purpose of highlighting features other

than connectivity have also been explored in the literature. For example, the modified forces of Noack (2007) and Agutter and Wheatley (2004) emphasise intra-cluster forces in order to separate groupings in small world graphs. The ImPrEd model of Simonetto et al. (2011) adds extra node–edge repulsive forces and constraints in order to improve a layout whilst preserving its planarity. Further details on model variants can be found in the surveys of Herman, Melançon, and Marshall (2000), Brandes (2001b), and von Landesberger et al. (2011)

The energy function that will become the focus for the rest of this chapter is known as *stress*, and was popularised within the context of graph layout in by Kamada and Kawai (1989). Its energy is defined as

$$\text{stress}(\mathbf{X}) = \sum_{\{i,j\}:i<j} w_{ij}(||\mathbf{X}_j - \mathbf{X}_i|| - d_{ij})^2 \tag{2.8}$$

where w_{ij} and d_{ij} are constants specific to each term in the summation. The intuition behind this formulation is that there are Hooke's law springs attached between all pairs of vertices, not just those connected by edges. Each of these springs is of natural length d_{ij} and of stiffness w_{ij}. In practice, d_{ij} is usually set to the shortest-path distance between vertices, and w_{ij} is almost always set to d_{ij}^{-2} in order to suppress the contribution from long-range springs that would otherwise obfuscate local detail (Brandes and Pich, 2008).

The derivative of this equation is non-linear, like in Equation (2.4) or in Fruchterman and Reingold (1991) and Frick, Ludwig, and Mehldau (1995), and so cannot be solved exactly using linear solvers like for Equations (2.2) or (2.6). However, stress as an energy function is known to produce high-quality layouts (Brandes and Pich, 2008). This is partly because it manages to avoid the *peripheral effect* (Hu, 2005), a common visual artifact of many force-directed methods where repulsive forces tend to push nodes towards the boundary of the layout. This can be seen in Fig-

ure 2.3 for Eades' method; the grid on the left bulges slightly, while the binary tree and Sierpiński triangle underrepresent local detail. Stress avoids this by forgoing repulsive forces entirely in Equation (2.8), as can also be seen in the bottom row in Figure 2.3.

Stress also does not require extra input parameters such as those in Equation (2.4) because d_{ij} and w_{ij} are both derived from the structure of the graph itself. However, it is known to be difficult to optimise due to an abundance of local minima (de Leeuw, 1988; Gansner, Koren, and North, 2004), and also does not scale well to larger graphs due to the quadratic number of terms in the summation (Brandes and Pich, 2008; Hu, 2005).

The subsequent content of this chapter will aim at addressing these issues through the application of an algorithm known as *stochastic gradient descent* (SGD) to minimise Equation (2.8). Before describing the algorithm itself however, a history of other methods used to minimise stress will first be outlined.

2.1.2 Multidimensional scaling

Equation (2.8) was first utilised in a domain unrelated to graphs, but still for the purpose of visualisation, known as *multidimensional scaling* (MDS). As the name implies, this involves taking high-dimensional data and scaling it down to fewer dimensions, usually to two or three.

The difficulty in this task can be illustrated through a simple example: the humble tetrahedron. The task at hand is to find an 'ideal' drawing of the tetrahedron, where 'ideal' is defined as having each of its edges drawn with equal length. When one tries to draw such a configuration on a piece of (two-dimensional) paper, it quickly becomes clear that it is not possible. Even for such a small graph with only four vertices, there are too few dimensions available to provide sufficient degrees

of freedom. The next logical question is: what layout gets as close as possible to this ideal?

Multidimensional scaling (MDS) is a technique to solve exactly this type of problem, that attempts to minimize the disparity between ideal and low-dimensional distances. This is done by defining an equation to *quantify* the error in a layout, and then minimizing it in a similar fashion to the algorithms above in Section 2.1.1. While this equation can come in many forms (Cox and Cox, 2000), stress as in Equation (2.8) is the most commonly used for graph layout (Brandes and Pich, 2008), for the reasons described at the end of the previous section.

The use of Equation (2.8) was popularized for graph layout by Kamada and Kawai (1989) who minimized the function using a localized 2D Newton-Raphson method, while within the MDS community Kruskal (1964a) originally used gradient descent (Kruskal, 1964b). This was later improved upon by de Leeuw (1988) with a method known as *majorization*, which minimizes a complicated function by iteratively finding the true minima of a series of simpler functions, each of which touches the original function and is an upper bound for it (Cox and Cox, 2000). This was applied to graph layout by Gansner, Koren, and North (2004) and has been the state-of-the-art for the past decade.

2.1.3 Majorization

Majorization is the algorithm that will be used throughout this chapter as the benchmark against which the performance of SGD will be assessed. It works by first finding a Laplacian matrix $\mathbf{L}^w$, defined as

$$\mathbf{L}^w_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ -\sum_{k \neq i} \mathbf{L}^w_{ik} & \text{if } i = j \end{cases} \tag{2.9}$$

and then another Laplacian matrix $\mathbf{L}^{\mathbf{X}}$, defined as

$$\mathbf{L}^{\mathbf{X}}_{ij} = \begin{cases} -w_{ij}d_{ij}||\mathbf{X}_i - \mathbf{X}_j||^{-1} & \text{if } i \neq j \\ -\sum_{k\neq i}\mathbf{L}^{\mathbf{X}}_{ik} & \text{if } i = j. \end{cases} \tag{2.10}$$

It can be shown that the equation

$$\mathbf{L}^{w}\mathbf{x}' = \mathbf{L}^{\mathbf{X}}\mathbf{x} \tag{2.11}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are column vectors that represent single dimensions of the full layouts $\mathbf{X}$ and $\mathbf{X}'$ respectively, guarantees that stress($\mathbf{X}'$) $\leq$ stress($\mathbf{X}$) (Gansner, Koren, and North, 2004). The optimisation process is then performed simply by solving Equation (2.11) as a system of linear equations, with $\mathbf{x}'$ as the column of unknowns being solved for. This is repeated in an iterative manner, where $\mathbf{x}'$ is computed in one iteration and then used as $\mathbf{x}$ in the next, until the algorithm can no longer improve the value of stress by more than a certain threshold. Note that $\mathbf{L}^{w}$ only needs to be computed once, but $\mathbf{L}^{\mathbf{X}}$ must be recalculated on every iteration.

Additionally in practice, the position of the first vertex is fixed to the origin, and the first row and column of $\mathbf{L}^{w}$ are removed, as well as the first row of the vector $\mathbf{L}^{\mathbf{X}}\mathbf{x}$. This is in order to remove translational invariance from the solution, which allows for fast linear equation solving methods to be applied to solve Equation 2.11. In particular, Gansner, Koren, and North (2004) recommend *Cholesky factorization* (Press et al., 2007c) or the *conjugate gradient* method (Press et al., 2007b), both of which will be tested later in Section 2.2.3.

Gansner, Koren, and North (2004) also described another *local* method for solving Equation 2.11, where the system of equations is solved for just one vertex at a

time. This is done by setting the position of vertex i to

$$\mathbf{x}_i' = \frac{\sum_{j \neq i} w_{ij}(\mathbf{x}_j + d_{ij}(\mathbf{x}_i - \mathbf{x}_j)||\mathbf{X}_i - \mathbf{X}_j||^{-1})}{\sum_{j \neq i} w_{ij}} \tag{2.12}$$

and is also guaranteed to monotonically decrease stress (Gansner, Koren, and North, 2004). This local method will also be tested later in Section 2.2.3, and will furthermore be important when considering large graphs in Section 2.3.

2.1.4 Constraint relaxation

The origin of the novel contributions in this chapter is rooted in constrained graph layout, where a relaxation algorithm, here referred to as *constraint relaxation*, has gained popularity due to its simplicity and versatility (Dwyer, 2009; Bostock, Ogievetsky, and Heer, 2011). This algorithm will provide the context to the geometric interpretation of SGD that will be leveraged to make the modifications in Section 2.2.

Constraint relaxation was first introduced in video game engines as a technique to quickly approximate the behavior of cloth, which is modeled as a planar mesh of vertices that maintains its edges at a fixed length. A full physics simulation would represent each edge as a stiff spring, summing up and integrating over the resulting forces, but a realistic piece of cloth contains too many edges for this to be feasible. To avoid this bottleneck, Jakobsen (2001) introduced the idea of considering each edge independently, moving a single pair of vertices at a time, as shown in Figure 2.4. While this is a rather simple and perhaps naive idea, in practice the solution converges in very few iterations.

This was utilized by Dwyer (2009), who used the method in conjunction with any standard force-directed layout to achieve effects such as making edges point down-

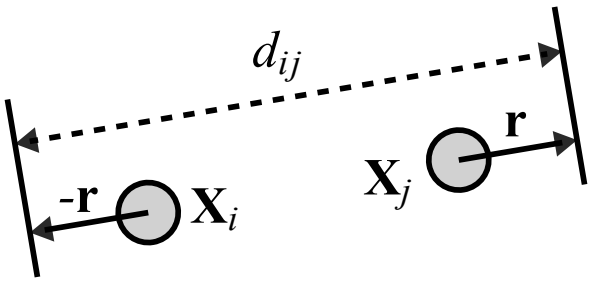


Figure 2.4: Satisfaction of the distance constraint described by Equation (2.13).

wards, or fixing cycles around the edge of a wheel. To define it properly in the case of maintaining a distance d_{ij} between the coordinates of two nodes $\mathbf{X}_i$ and $\mathbf{X}_j$, this *constraint* can be written as

$$||\mathbf{X}_i - \mathbf{X}_j|| \leftarrow d_{ij} \tag{2.13}$$

and is *satisfied* by moving $\mathbf{X}_i$ and $\mathbf{X}_j$ in opposite directions by a vector

$$\mathbf{r} = \frac{||\mathbf{X}_i - \mathbf{X}_j|| - d_{ij}}{2} \frac{\mathbf{X}_i - \mathbf{X}_j}{||\mathbf{X}_i - \mathbf{X}_j||}. \tag{2.14}$$

This can be seen as a diagram in Figure 2.4, and is analogous to decompressing an infinitely stiff spring of length d_{ij}.

Rewriting Equation (2.8) as

$$\text{stress}(\mathbf{X}) = \sum_{i<j} Q_{ij}(\mathbf{X}), \tag{2.15}$$

$$Q_{ij}(\mathbf{X}) = w_{ij}(||\mathbf{X}_i - \mathbf{X}_j|| - d_{ij})^2, \tag{2.16}$$

it can be seen that if every term Q_{ij} in the summation is satisfied as a constraint (2.13), then the total stress is zero, corresponding to an ideal layout. This is exactly the idea behind the method that will be studied here – a constraint is placed on every possible pair of vertices, and then satisfied one by one as above.

However zero stress is almost always impossible, for the same reasons that the aforementioned tetrahedron cannot be embedded in 2D. In such situations, simply satisfying constraints does not lead to convergence, but the work in this chapter will be dedicated to describing a simple extension that not only converges but, remarkably, also minimises stress. This extension can be found from Section 2.2 onwards.

2.1.5 Large graphs

To understand how many layout algorithms tackle scaling to larger graphs, it is useful to rewrite Equation (2.8) by splitting the summation into two parts: paths that traverse one edge, and paths that traverse multiple. This can be written as

$$\text{stress}(\mathbf{X}) = \sum_{\{i,j\}\in E} w_{ij}\sigma_{ij} + \sum_{\{i,j\}\notin E} w_{ij}\sigma_{ij} \tag{2.17}$$

where $\sigma_{ij} = (||\mathbf{X}_i - \mathbf{X}_j|| - d_{ij})^2$. Just considering the preprocessing stage for now, it is clear that d and w can be easily computed for the left half of the summation directly from the graph. Real-world graphs are also usually sparse, so for a graph with $|V|$ vertices and $|E|$ edges, $|E| \ll |V|^2$ making the space required to store these values tolerable. However the second half is not so easy – an all-pairs shortest paths (APSP) calculation takes $O(|E| + |V|)$ time per vertex for an unweighted graph with a breadth-first search, or $O(|E|+|V| \log |V|)$ for a weighted graph using Dijkstra's algorithm (Cormen et al., 2009).

The second stage is iteration, where the layout is gradually improved towards a good minimum. Again, computing the first summation is tolerable, but the number of longer distance contributions quickly grows out of control. Many notable attempts have been made at tackling this second half. A common approach is to

ignore d_{ij}, and to approximate the summation as an n-body repulsion problem, which can be efficiently well approximated using k-d trees (Barnes and Hut, 1986). Hu (2005) and independently Hachul and Junger (2004) used similar tricks in the context of the force-directed model of Fruchterman and Reingold (1991), along with a multilevel coarsening scheme to avoid local minima. Multilevel methods will not be considered here, but a comparison and review can be found in Bartel et al. (2010).

Brandes and Pich (2008) even ignore the second half completely and capture the long-range structure by first initializing with a fast approximation to classical scaling known as PivotMDS (Brandes and Pich, 2007), which minimizes the inner product rather than Euclidean distance. This idea of smart initialisation will be the subject of Section 2.2.4.

There are a couple of issues with this idea, one being that treating all long-range forces equally is unfaithful to graph-theoretic distances, and another being that the relative strength of these forces depends on an extra parameter that can greatly affect the final layout of the graph (Hu, 2005). Keeping these dependent on their graph-theoretic distance sidesteps both of these issues, but brings back the problem of computing and storing shortest paths. One approach to maintaining this dependence comes from Khoury et al. (2012), who use a low-rank approximation of the distance matrix based on its singular value decomposition. This can work extremely well, but still requires APSP unless $w_{ij} = d_{ij}^{-1}$.

Within the normal MDS community, there have been multiple attempts to approximate stress, such as Halko, Martinsson, and Tropp (2011) or GLINT (Ingram and Munzner, 2012). However an additional problem with graphs is that shortest paths still need to be computed, and so even the distances between datapoints are not immediately available. This is an assumption that these approximations usually have, and so they cannot be directly used in the contexts of graphs. Fortu-

nately, Ortmann, Klimenta, and Brandes (2017) developed an approach designed around graphs. They pick a set of 'pivot' vertices whose shortest paths are used as an approximation for the shortest paths of vertices close to them. Since this approach reduces the number of terms in the summation, using it in the context of SGD also reduces the amount of work per iteration. The use of this sparse approximation will be explored in Section 2.3. It will be leveraged in order to scale the contributions outlined in the following sections up to large graphs.

2.2 Stochastic gradient descent

The remaining content within this chapter will describe the novel application of stochastic gradient descent (SGD) to minimise stress. SGD is a method widely used in other fields, most notably in machine learning (Bottou, 2012), but was not previously applied to minimise Equation 2.8. SGD approximates the gradient of a complex function using an unbiased estimator of that function, and in the case that the complex function is a sum, the estimator may use the gradients of its individual terms (Bottou, 2012).

For stress in particular, this has an intuitive geometric interpretation of moving a single pair of vertices at a time; this interpretation allows for a simple modification to the optimisation step which help to avoid local minima and speed up convergence. The benefits of SGD over majorization will be shown through experiment.

The modifications required to the constraint relaxation previously described in Section 2.1.4 can be understood by first noticing that satisfying a constraint is equivalent to moving both vertices in the direction of the gradient of a stress term Q_{ij}

$$\frac{\partial Q_{ij}}{\partial \mathbf{X}_i} = \frac{\partial}{\partial \mathbf{X}_i} w_{ij}(||\mathbf{X}_i - \mathbf{X}_j|| - d_{ij})^2 = 4w_{ij}\mathbf{r}. \tag{2.18}$$

The full gradient $\partial Q_{ij}/\partial \mathbf{X}$ can be written as

$$\frac{\partial Q_{ij}}{\partial \mathbf{X}_k} = \begin{cases} 4w_{ij}\mathbf{r} & \text{if } k = i \\ -4w_{ij}\mathbf{r} & \text{if } k = j \\ 0 & \text{otherwise.} \end{cases} \tag{2.19}$$

Recall that standard force-directed methods use (non-stochastic) gradient descent, following Equation (2.5), and that this involves taking a step in the opposite direction to the derivative of the entire summation at once. The change required for stochastic gradient descent is as simple as doing the same thing, but to a single term at a time.

Specifically, this involves repeatedly selecting a single term Q_{ij} and applying the iterative formula $\mathbf{X} \leftarrow \mathbf{X} - \eta \nabla Q_{ij}(\mathbf{X})$. Note that since this gradient is zero with respect to all $\mathbf{X}_k$ other than $\mathbf{X}_i$ and $\mathbf{X}_j$, it suffices to update the positions of $\mathbf{X}_i$ and $\mathbf{X}_j$ by

$$\begin{bmatrix} \mathbf{X}_i \\ \mathbf{X}_j \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{X}_i \\ \mathbf{X}_j \end{bmatrix} + \begin{bmatrix} \Delta\mathbf{X}_i \\ \Delta\mathbf{X}_j \end{bmatrix} = \begin{bmatrix} \mathbf{X}_i \\ \mathbf{X}_j \end{bmatrix} - 4w_{ij}\eta \begin{bmatrix} \mathbf{r} \\ -\mathbf{r} \end{bmatrix}. \tag{2.20}$$

The constraint relaxation of the previous section is therefore equivalent to a special case of SGD where $w_{ij} = 1$ and $\eta = 1/4$.[3] Additionally in this case, a gradual reduction in the step size η is necessary to make the process converge, since not all terms can be simultaneously minimised. This will be further elaborated in Section 2.2.1, and the problem can be interpreted in the context of stress as each edge of the aforementioned tetrahedron satisfying its constraint one by one as in Figure 2.4, but never finding a configuration where all are satisfied at the same time.

[3] This equivalence to stochastic gradient descent was not known for a long time, as neither Jakobsen (2001) nor Dwyer (2009) classified the algorithm correctly. In fact, not even my supervisors and I knew that this was SGD in the first version of this paper we submitted to a journal, where we chose the name *weighted constraint relaxation*.

Writing $\mu = 4w_{ij}\eta$ as the coefficient of $\mathbf{r}$, it can be seen that $Q_{ij} \leftarrow 0$ when $\mu = 1$ and decreases monotonically from $\mu = 0$ to $\mu = 1$. In other words, the exact step size required to optimally minimise each term Q_{ij} can always be instantly known. This is the geometric interpretation that allows for a modification to SGD, specific to the context of stress. It involves setting a hard upper limit of $\mu \leq 1$:

$$\begin{aligned} \Delta\mathbf{X}_i = -\Delta\mathbf{X}_j &= -\mu\,\mathbf{r}, \\ \mu &= \min\{\, w_{ij}\eta,\, 1 \,\} \end{aligned} \tag{2.21}$$

where the constant factor of 4 has been absorbed into η for brevity. This modified algorithm makes updates that are identical to standard SGD when η is sufficiently small, at

$$\eta < \frac{1}{w_{\max}}. \tag{2.22}$$

Since this will always eventually be the case because η tends to zero, it has the same asymptotic convergence properties as standard SGD, which will be discussed in Section 2.2.1.

Introducing this upper limit on μ allows for much larger initial step sizes than standard SGD, yielding much faster convergence without needing to worry about divergence due to exploding gradients (Goodfellow, Bengio, and Courville, 2016). It will be shown by experiment that this results in state-of-the-art performance for a wide range of graphs (except for a single specific case, see Section 2.2.3). In addition, *random reshuffling* of terms is used unless otherwise stated; see Section 2.2.2 for further elaboration. A full pass through all the terms Q_{ij} will be referred to as an *iteration*, while a single application of Equation (2.20) will be referred to as a *step*. From now on, the modified SGD algorithm simply as SGD.

A crucial aspect of the experiments performed here is that vertex positions were initialised uniformly randomly within a 1×1 square, whereas a common first step to

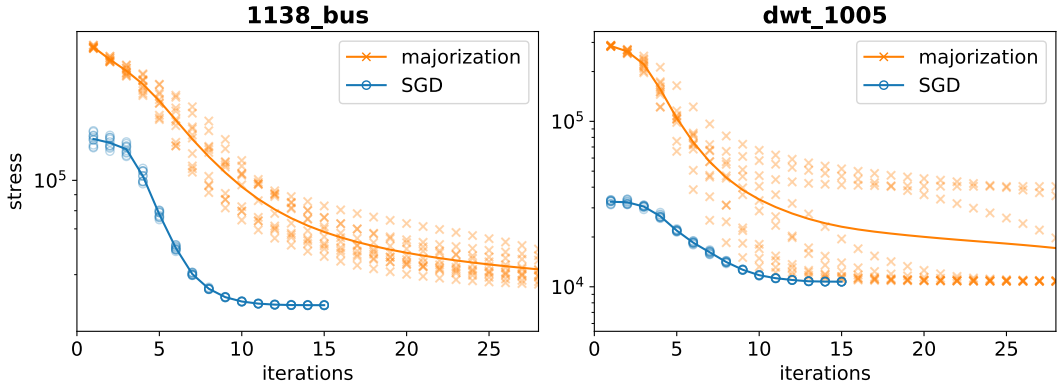


Figure 2.5: Plots of stress for SGD and majorization on the graphs `1138_bus` and `dwt_1005`, each initialized randomly within a 1×1 square. The circles and crosses show stress on each iteration over 10 runs, with the line running through the mean. Initial stress values are omitted. SGD is clearly more consistent, always reaching lower stress levels than majorization ever manages in hundreds of iterations on `1138_bus`. They both reach the same overall minimum on the more mesh-like `dwt_1005`, but majorization often gets stuck on a particularly dangerous local minimum, shown by its diverging paths. A more detailed timing analysis on a wide variety of other examples can be seen in Section 2.2.3.

Algorithm 1: Stochastic Gradient Descent

1 **SGD** (G)**:**
  inputs : graph $G = (V, E)$
  output: k-dimensional layout $\mathbf{X}$
2 $d_{ij} \leftarrow$ SHORTESTPATHS(G)
3 $\mathbf{X} \leftarrow$ RAND($|V|, k$)
4 **for** η in ANNEALINGSCHEDULE() **:**
5 **foreach** $\{i, j\} : i < j$ in random order **:**
6 $\mu \leftarrow w_{ij}\eta$
7 **if** $\mu > 1$ **:**
8 $\mu \leftarrow 1$
9 $\mathbf{r} \leftarrow \frac{\|\mathbf{X}_i - \mathbf{X}_j\| - d_{ij}}{2} \frac{\mathbf{X}_i - \mathbf{X}_j}{\|\mathbf{X}_i - \mathbf{X}_j\|}$
10 $\mathbf{X}_i \leftarrow \mathbf{X}_i - \mu\,\mathbf{r}$
11 $\mathbf{X}_j \leftarrow \mathbf{X}_j + \mu\,\mathbf{r}$

Figure 2.6: Pseudocode for the algorithm described in Section 2.2. The results in this paper initialize positions randomly within a 1×1 square on line 3. The annealing schedule on line 4 is explained in Section 2.2.1.

constructing a node layout is to instead initialise using a fast linear method, such as PivotMDS (Brandes and Pich, 2007). The inclusion of this extra initialisation step brings the performance of majorization level to SGD in terms of final stress reached, and will be further discussed in Section 2.2.4.

Plots of stress achieved using SGD compared to majorization are presented briefly in Figure 2.5, and in more detail in Section 2.2.3. Pseudocode is shown in Algorithm 1, Figure 2.6. All results were performed using C# running in Visual Studio, on an Intel Core i7-4790 CPU with 16GB of RAM. Unless stated otherwise, graph data is from the SuiteSparse Matrix Collection (Davis and Hu, 2011).

2.2.1 Step size annealing

The process of gradually reducing the step size over the course of the optimisation, so that the process can converge, will be referred to as an *annealing schedule*. Choosing a good annealing schedule η is crucial to the performance of SGD in all contexts (Darken, Chang, and Moody, 1992), and a typical implementation can involve complex algorithms for tuning the step size to the problem at hand (Ruder, 2016). Most of these methods do not apply here for two reasons. First, due to the limit on the step size in Equation (2.21), much larger step sizes than standard SGD would allow can and will be used. Second, many of these methods use previous gradients to inform the step size; only positions of the two vertices directly involved are updated, so storing and applying previous gradients is inefficient to the point of increasing the asymptotic complexity of the algorithm.

Even ignoring such adaptive methods, the full (infinite) space of possible annealing schedules is too large to investigate in its entirety, and results can even differ depending on the input graph. A limited subset of possible schedules will therefore be tested, taking the mean final stress across a wide range of graphs as the

performance criterion (the full set of graphs considered in Section 2.2.3). Two use cases will be considered: one where time is a limiting factor and so the number of iterations is fixed, and another where the algorithm may continue until the layout has converged to within a desired accuracy.

Fixed number of iterations

The schedules to be studied here will consider step size that starts at a maximum value $\eta = \eta_{\max}$ at the first iteration $t = 0$, and decreases monotonically to $\eta = \eta_{\min}$ at the final iteration $t = t_{\max} - 1$. Large values of η result in all μ capped at 1, and very small values will result in little to no movement of vertices. Because the useful range exists only in between these extremes, η is set to

$$\eta_{\max} = \frac{1}{w_{\min}}, \quad \eta_{\min} = \frac{\varepsilon}{w_{\max}}. \tag{2.23}$$

In this case $w_{ij} = d_{ij}^{-2}$ so $w_{\min}$ is inversely proportional to the diameter of the graph $d_{\max}$, and $w_{\max}$ to the smallest edge length $d_{\min}$. This choice of $\eta_{\max}$ ensures that all $\mu = 1$ for the first iteration, resulting in the constraint relaxation described in Section 2.1.4 applied to all pairs of vertices. The choice of $\eta_{\min}$ ensures that even the strongest constraints reach a small value of $\mu = \varepsilon$ for the final iteration.

The performance was computed for various schedules $\eta(t)$ where t is the iteration number, constrained to $\eta(0) = \eta_{\max}$ and $\eta(t_{\max} - 1) = \eta_{\min}$ (except for the special case $\eta(t) = 1$). In each panel of Figure 2.7 the form of the function $\eta(t)$ is varied for a fixed choice of $t_{\max}$ and ε, on various input graphs. `1138_bus` shows the typical behavior of $\eta = ae^{-bt}$ reaching lower stress and $\eta = a/(1 + bt)$ never quite catching up; `lesmis` shows that this applies to smaller graphs as well; `dwt_1005` emphasizes the importance of larger step sizes, as the constant $\eta = 1$ struggles to ever jump over large local minima. Note that on this easier graph $\eta = a/(1 + bt)$

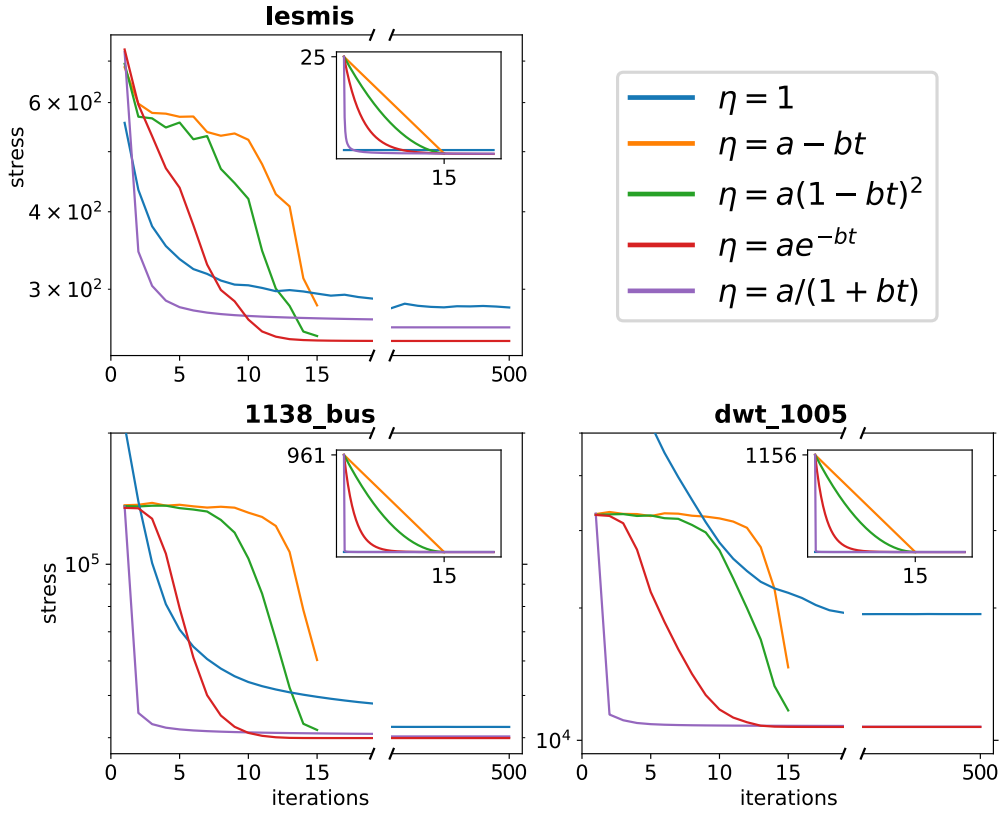


Figure 2.7: Plots of mean stress against iterations over 25 runs for the annealing schedules discussed in Section 2.2.1, with $t_{\max} = 15$ and $\varepsilon = 0.1$ in all cases. The exact schedules used are shown inset in the top right of every plot. To approximate behavior given unlimited time, schedules were run for 500 iterations.

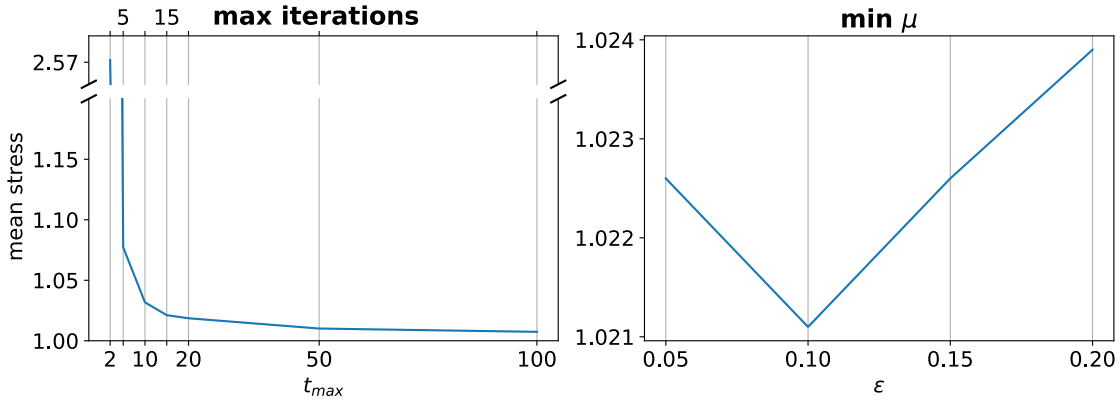


Figure 2.8: Plots of mean stress over 25 runs on all graphs in Section 2.2.3 when varying the parameters $t_{\max}$ or ε on Equation (2.24), normalized to the best values over all runs from Figure 2.10. There are clear diminishing returns when increasing $t_{\max}$, so $t_{\max} = 15$ was chosen as a trade-off between speed and quality. $\varepsilon = 0.1$ is close to optimum for this value.

is enough to reach good minima in very few iterations.

The best form of $\eta(t)$ appears to be the exponential decay given by the equation

$$\eta_1(t) = \eta_{\max} e^{-\lambda t}. \tag{2.24}$$

In addition, the parameters $t_{\max}$ and ε where varied for this form of $\eta(t)$ (see Figure 2.8). Increasing $t_{\max}$ always improves the quality but also increases computation time, so $t_{\max} = 15$ was chosen as a reasonable compromise between speed and quality. With this number of iterations most of the gains had already been made and further ones gave diminishing returns, although for particular applications another choice may be more appropriate. The right-hand side of Figure 2.8 shows that the choice $\varepsilon = 0.1$ appears to be close to optimal for this $t_{\max}$.

It is common in SGD to use a schedule $\eta = \Theta(1/t)$ (Darken, Chang, and Moody, 1992), however for the small number of iterations considered here, the large initial step sizes cause η to decay too quickly in the beginning, leading to worse local minima. Exponential decay drops faster than $1/t$ as $t \to \infty$, but $1/t$ drops faster in early iterations given fixed values at $\eta(0)$ and $\eta(t_{\max} - 1)$, as shown by the inset panels in Figure 2.7. An exponential schedule is also commonly used in simulated annealing (Davidson and Harel, 1996) a technique with similarities to gradient and stochastic gradient descent (Pirlot and Vidal, 1996).

Unlimited iterations

The schedule described above works well in practice for a fixed number of iterations, but given more time it can be desirable to let the algorithm run for longer to produce an optimal layout. Here a schedule will be described that is guaranteed to converge, with a stopping criterion to prevent the algorithm from wasting iterations on negligible movements.

A proof of convergence for SGD is well known in the machine learning literature (Bottou, 2012), and requires an annealing schedule that satisfies

$$\sum_{t=0}^{\infty} \eta(t) = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \eta(t)^2 < \infty. \tag{2.25}$$

This is guaranteed to reach global minima under conditions slightly weaker than convexity (Bottou, 1998). Intuitively, the first summation ensures the decay is slow enough to reach the minimum no matter how far away it is initialized, and the second ensures fast enough decay to converge to, rather than bounce around the minimum (Welling and Teh, 2011). In the context of non-convex functions, like the stress equation considered in this paper, such a proof only holds for convergence to a stationary point that may be a saddle (Bottou, 1998). There is also recent work proving convergence to local minima in specific classes of non-convex functions (Ge et al., 2015).

Since a global minimum cannot be guaranteed with any choice of schedule, the best that can be done is to choose a schedule that will converge to a stationary point. A commonly used schedule that guarantees this is $\eta = \Theta(1/t)$, as this satisfies Equation (2.25). However, in the previous section it was noted that this schedule gets stuck in poor local minima. A mixed schedule can therefore be used: when t is small, $\eta(t) = \eta_1(t)$ follows the exponential schedule of the previous section, because in practice this avoids poor local minima; when t is large this is switched to a $1/t$ schedule to guarantee convergence to a stationary point, following

$$\eta_2(t+\tau) = \frac{w_{\max}^{-1}}{1+\lambda t} \quad \text{when} \quad t > \tau \;:\; \eta_1(\tau) = w_{\max}^{-1}. \tag{2.26}$$

The cross-over value τ is the iteration at which the limit in Equation (2.21) stops capping μ and the algorithm becomes standard SGD. Since there are now more iterations to work with, $t_{\max} = 30$ is chosen in order to further improve avoid-

ance of local minima. This choice is sufficient to give even or better mean performance than majorization after convergence across every graph tested except for one (see Section 2.2.3), but again depending on the application another choice may be more suitable.

Finally, a suitable stopping criterion is needed to avoid the algorithm from never stopping, although in practice a hard upper limit for t is also included. Since SGD does not guarantee the monotonic decrease of stress (Darken, Chang, and Moody, 1992), the majorization heuristic adopted by Gansner, Koren, and North (2004) cannot be used, which stops when the relative change in stress drops below a certain threshold.

However it is guaranteed that each time a constraint is satisfied, its corresponding term within the summation does decrease. How close the algorithm is to convergence is therefore estimated by tracking the maximum distance any vertex is moved by a single step over the previous iteration. The optimisation is stopped when this crosses a threshold

$$\max ||\Delta \mathbf{X}|| < \delta. \tag{2.27}$$

A value of $\delta = 0.03$ works well in practice, and is used for the results in Section 2.2.3.

Thus two schedules have been designed: one for a fixed number of iterations, and one that continues until convergence. Results using both of these are presented in Section 2.2.3. It is important to note that these schedules use simple heuristics, and the exact nature of the data will affect the results. However they are robust across a wide variety of graphs, as all the results shown in this paper use these two schedules.

2.2.2 Randomisation

An important consideration is the order in which constraints are satisfied, as naive iteration can introduce biases that cause the algorithm to get caught in local minima. The original method behind SGD, proposed by Robbins and Monro (1951), randomizes with replacement, meaning that a random term is picked every time with no guarantee as to how often a term will be picked. Some variants perform random reshuffling (RR) which guarantees that every term is processed once on each iteration. Under certain conditions it can be proven analytically that RR converges faster (Gürbüzbalaban, Ozdaglar, and Parrilo, 2019), and the results here support this.

Unfortunately adding randomness incurs a penalty in speed, due to the cost of both random number generation and reduced data cache prefetching. This overhead is non-trivial, with iterations taking up to 60% longer with random reshuffling compared to looping in order. The trade-offs between more randomness for better convergence but slower iterations, versus less randomness for slower convergence but faster iterations, will therefore be explored here. Five different degrees of randomness were tried: shuffling only the indices themselves, which removes any bias inherent to the data but still makes use of the cache by iterating in order; randomizing with replacement; shuffling the order of terms once; shuffling twice and alternating between the two orders; and shuffling on every iteration. The results can be seen in Figure 2.9.

Five different graphs were selected, each with around 1000 vertices, and show a corresponding good layout for each to visualize the differences between them. More mesh-like graphs such as `dwt_1005` do not benefit much from added randomness, and receive large gains in speed for a small hit to quality. As graphs get more difficult to draw, shuffling only indices quickly becomes ineffective, with

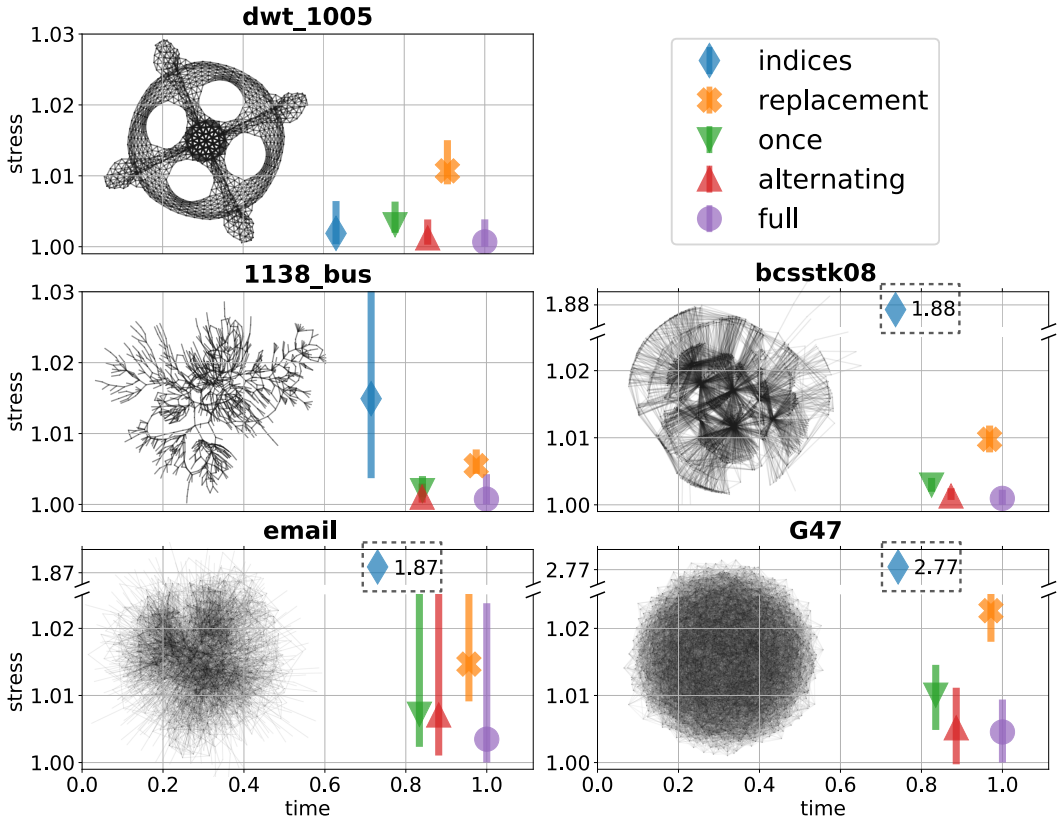


Figure 2.9: Plots of stress against time, taken using different degrees of randomization, over 50 runs from different random starting configurations. Markers indicate mean stress, with vertical error bars ranging from best to worst over all runs. Graphs are arranged in order of mean stress, with both stress and time normalized to the absolute minimum and maximum respectively over any run.

mean stress levels off by orders of magnitude on the plots with broken axes. The graph `email` is a social network, which tend to be very difficult to draw as their global minima can be hard to find. The drop in quality when reducing the randomness reflects this. `G47` is a random graph and has the highest stress, but is easier to draw since there are many minima close to global that are all relatively easy to find. In other words, even a random layout would achieve a decent value for stress here.

Although RR is the most expensive method, it is only slightly more expensive and consistently performs best. However if speed is the most important concern, alternating between two random shuffles gives stress levels that are in many cases almost as good, at a slightly reduced cost. RR is used for the rest of the results here.

2.2.3 Systematic study

To test the effectiveness of SGD, symmetric sparse matrices from the SuiteSparse Matrix Collection (Davis and Hu, 2011) will be used as a benchmark, following Khoury et al. (2012). Both SGD and majorization were ran on every graph with 1000 or fewer vertices, and compared the range of stress levels reached after 15 iterations and until convergence, using the two schedules described Section 2.2.1. These results can be seen in Figure 2.10. Each graph in the collection is also assigned to a group as metadata, and graphs that share a group tend to have similar topologies; groups are kept together, ordered alphabetically, and within each group sorted by difference in mean stress when allowing for convergence. Thus both plots have the same order. Groups are demarcated by the alternating shaded background.

A representative selection of larger graphs were also chosen for more detailed timing results, showing multiple implementations of majorization and the time course of convergence, which can be seen in Figure 2.13.

Quality

Figure 2.10 shows that that SGD reaches the same low stress levels on almost every run for both annealing schedules. While majorization is proven to monotonically decrease stress (Gansner, Koren, and North, 2004), it can often struggle with local minima. This can be clearly seen in Figure 2.11, as majorization consistently shows larger variance in its stress trajectories from different starting configurations.

The layouts displayed in Figure 2.10 were chosen to highlight the effects of different types of graphs. From left to right, top then bottom: `G15` is a random graph with nodes decreasing in mean degree. These random graphs reach consistent

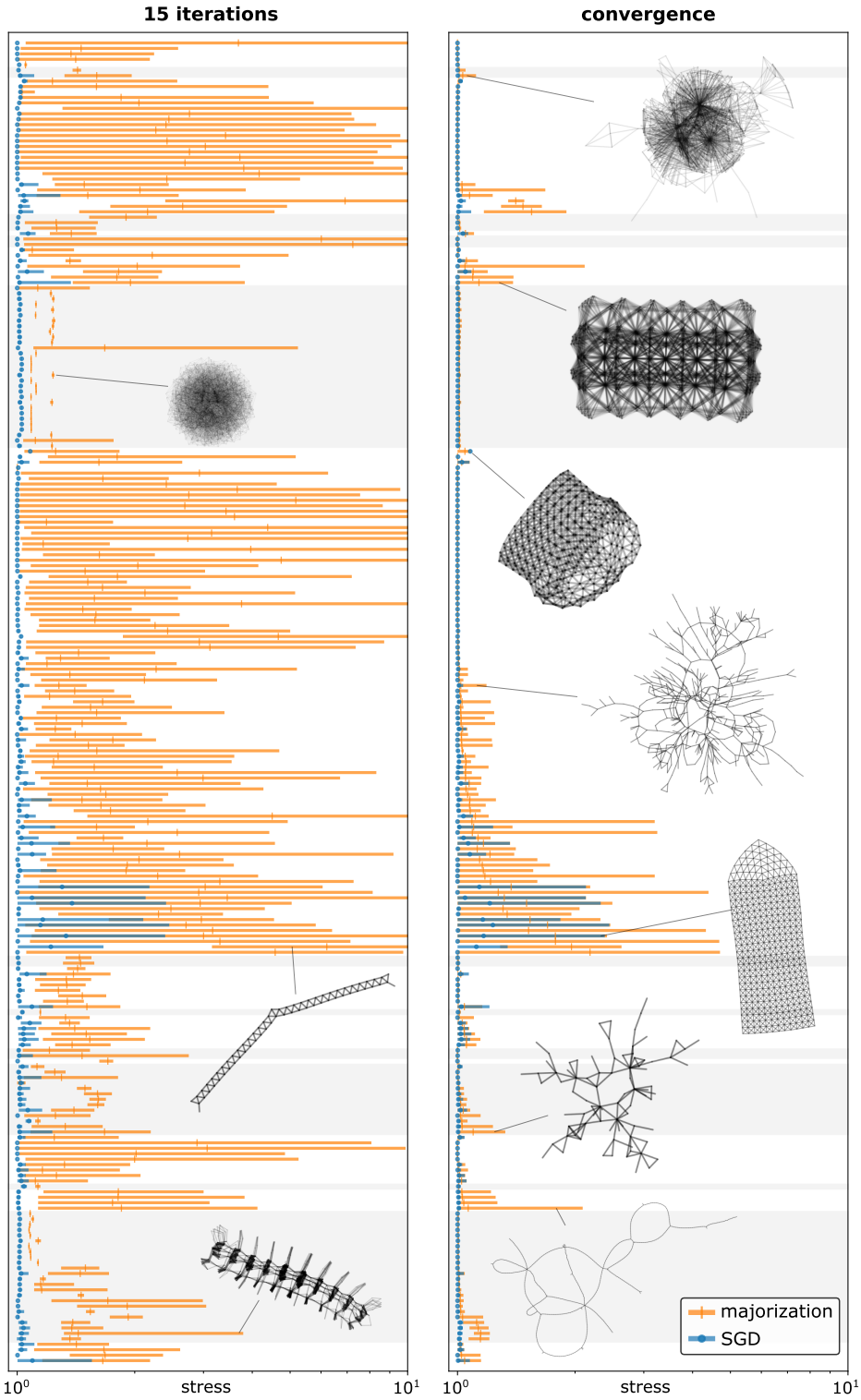


Figure 2.10: Stress achieved over 25 runs on 243 graphs in total. Markers indicate the mean over all runs, with bars ranging from minimum to maximum on any run. The left plot shows values reached after 15 iterations, and the right after convergence. Stress values are normalized to the lowest value achieved on all runs for either algorithm, as a baseline 'correct' layout. Layouts shown are described in Section 2.2.3. An animated version of the left plot can be viewed at www.youtube.com/watch?v=uv6Vw36KZ0k.

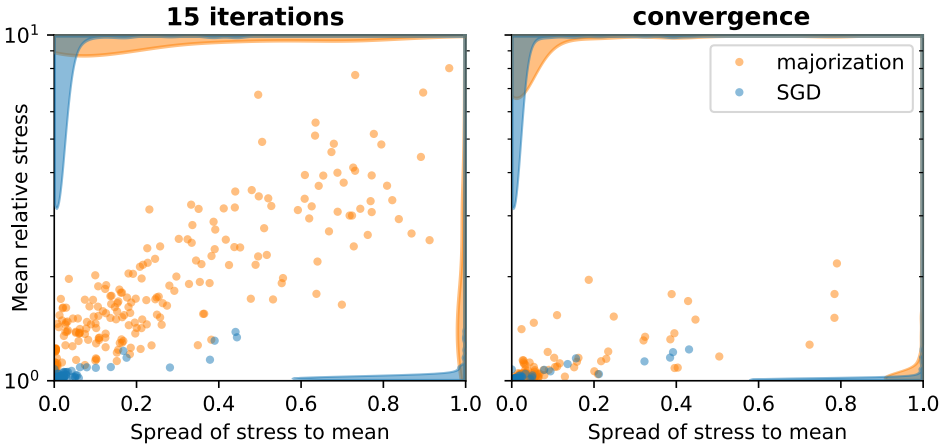


Figure 2.11: Scatter plots of mean stress relative to the best achieved, against the spread of the stress, measured as the coefficient of variation (standard deviation over mean), using the same results as Figure 2.10. The shaded regions on the top and right of each panel show the density of the mean (right) and spread (top) of the stress, computed using the `gaussian_kde` function of SciPy (Virtanen et al., 2020).

stress levels with both algorithms, as their lack of structure results in many minima close to global. `dwt_66` is an example of a graph that majorization struggles with, as it is very long and often has multiple twists that majorization cannot unravel. `orbitRaising_2` is a similar example, but in this case majorization also never reaches the global minimum, even after convergence. `celegans_metabolic` is a metabolic pathway that is around as densely packed as a graph worth drawing gets. SGD consistently outperforms majorization here too. Many of the largest ranges in the plot are from graphs similar to `ex2`; grids are difficult to fully unfold, and majorization often struggles with their many local minima.

On the other hand, `dwt_307` is the one graph (of the 243 investigated) where majorization reaches lower stress than SGD as a result of the modifications to standard SGD (see Figure 2.12).

`494_bus` is an example of the type of graph where stress as an energy function produces better layouts than other popular models. Its symmetry is clear here, whereas other force-directed algorithms can fail to show this due to the *periph-*

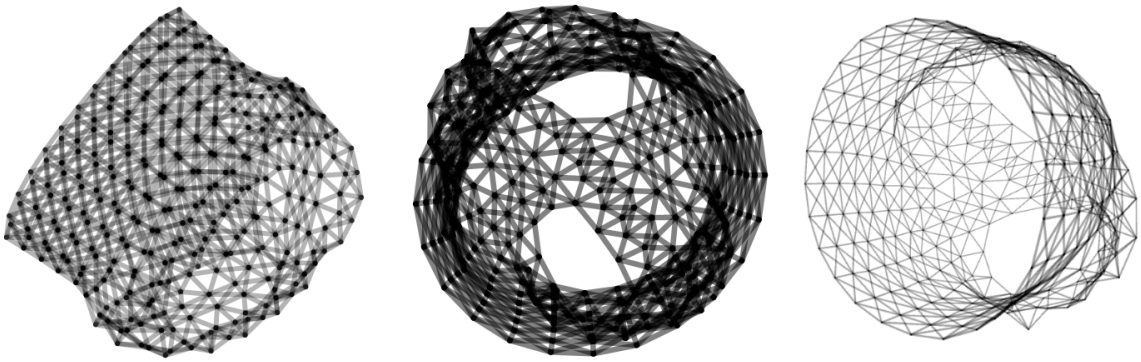


Figure 2.12: Layouts from `dwt_307`, the only one of the 243 graphs considered where majorization (left) yields lower stress than SGD (middle). The cylindrical shape of the graph is illustrated in a view of a 3D layout and projection on the right, which looks similar to if the middle layout were to be rotated with a positive yaw motion.

eral effect (Hu, 2005). `dwt_361` is an example of the type of graph that both SGD and majorization struggle with: long graphs that can twist. A twist in a graph constitutes a deep local minimum that iterative methods struggle with in general, and SGD is still susceptible to this issue.

`Sandi_authors` is a small graph, but with some densely packed sections that can become stuck behind each other, something that majorization often struggles with. And finally, `S10PI_n1` is a long graph that does not get twisted and so SGD deals with it perfectly well, but its long strands still tend to give majorization problems.

Speed

SGD converges to low stress levels in far fewer iterations than majorization. Graphs are laid out in only 15 iterations in the left plot in Figure 2.10, and there is not much improvement to be gained from using the convergent schedule to let the algorithm run for longer. This indicates that most global minima can be found in very few iterations, making SGD especially suited for real-time applications such as interactive layout. The stopping criterion used for majorization was for rela-

tive decrease in stress to be less than 10^{-5}, which is ten times more forgiving than originally suggested by Gansner, Koren, and North (2004), as 10^{-4} was not lenient enough to be confident that it had settled completely. Given enough time, majorization does almost always eventually find good minima, but can still settle in suboptimal minima and in some cases never finds the best configuration regardless of initialization. Majorization also takes many more iterations to converge than SGD, with means of 237 and 106 iterations respectively.

The real-world time per iteration must also be considered, even though both share a complexity $O(|V|^2)$. The Cholesky factorization routine from Numerical Recipes (Press et al., 2007c) was adapted to C#, and found that iterations are around 40% faster than SGD. However the initial decomposition before back-substitution requires $|V|^3/6$ iterations involving a multiply and a subtract (Press et al., 2007c), so the total time quickly tips in favor of SGD. Conjugate gradient (CG), with tolerance 0.1 and max iterations 10 as in Gansner, Hu, and North (2013), is an iterative method itself to solve the majorizing function and so iterates slower than Cholesky and SGD, but often beats out Cholesky overall when fewer iterations are necessary. CG and Cholesky also both benefit from optimized matrix multiplication routines (Gansner, Koren, and North, 2004) that were not tried here.

Localized majorization, which is also used to for majorization in the sparse model to be studied in Section 2.3, iterates fastest of all but converges slowest. It is also worth noting that over-shooting has been used before in the context of majorization to achieve an average of 1.5 times speedup (Wang and Wang, 2012).

Plots of stress against real time can be seen in Figure 2.13, where initial stress values are omitted, which is the cause of the horizontal offset to Cholesky due to its longer initialization time. Note that if the time for the first iteration for Cholesky is removed, it outperforms the other implementations of majorization, but still does not ever drop below the curve for SGD.

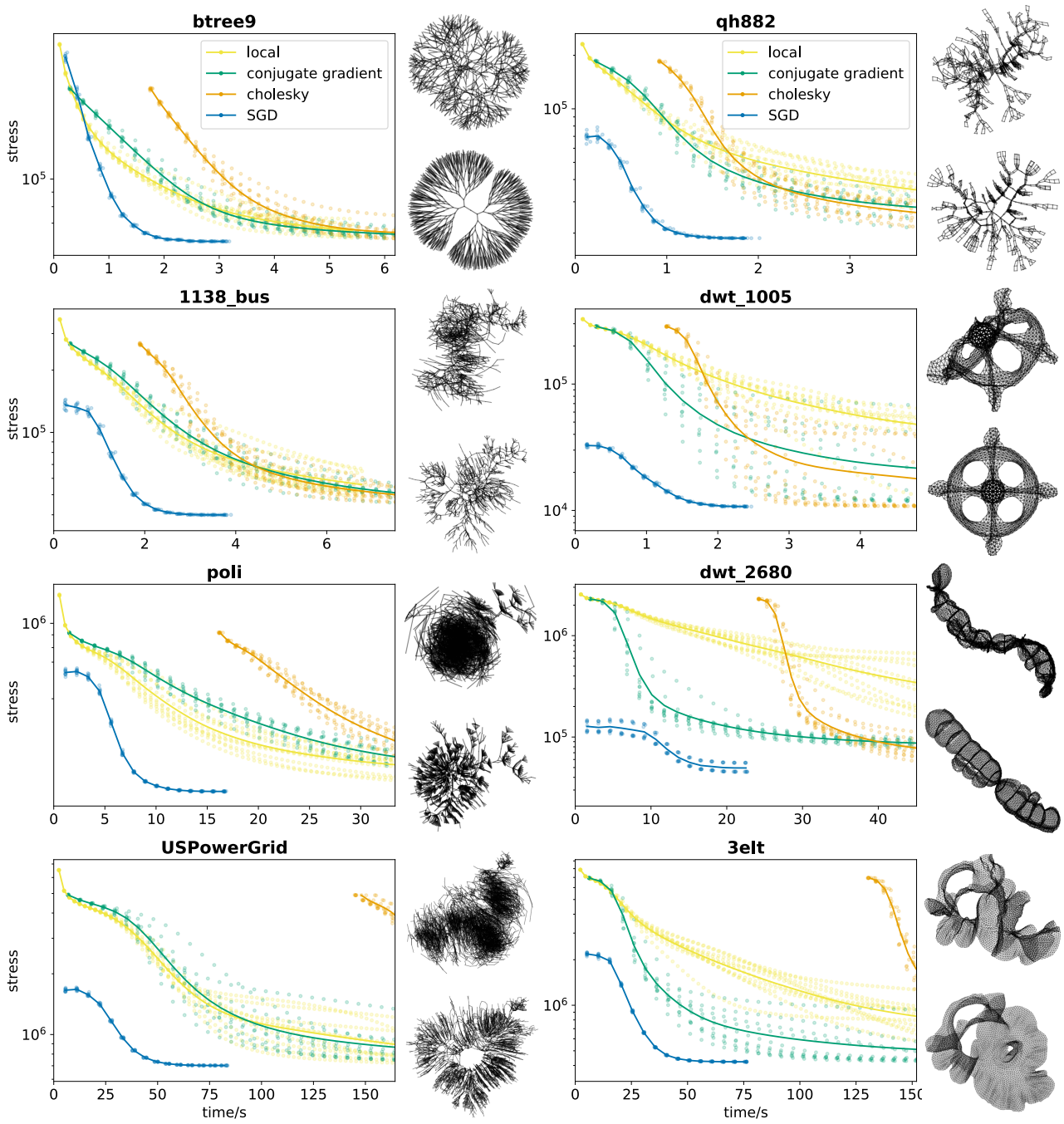


Figure 2.13: Graphs of stress against time for all three implementations of majorization, and SGD using the 15 iteration annealing schedule. 10 runs were used for each method, and the line plots run through the mean stress and time per iteration. Graphs were considered as unweighted and layouts were initialized randomly within a 1×1 square. The graph `btree9` is a binary tree of depth 9. The layouts show examples of what the graphs look like after 15 iterations, with Cholesky on top and SGD on bottom.

2.2.4 Smart initialisation

The results presented from the experiments performed here were recently reproduced and extended in the work of Börsig, Brandes, and Pasztor (2020). Specifically, they compared the performance of SGD against the localised version of majorization (see Equation (2.12)), with a focus on comparing the random initialisation used here against initialising with PivotMDS (Brandes and Pich, 2007).

Performance (measured in terms of minimum stress) given random initialisation was found to confirm the benefits of SGD presented here. Crucially however, the difference in performance when both methods are initialised with PivotMDS is negligible. Note that Börsig, Brandes, and Pasztor (2020) did not compare running times in their results, but noted that differences would likely depend more on implementation details than algorithmic differences, as both versions take a similar number of iterations to converge when given the smart initialisation.

It was also noted that SGD may be more amenable to parallelisation due to each step only moving two vertices at a time, a fact utilised by Olauson, Marin, and Söder (2020) in their parallel implementation of the method developed here.

The resilience of SGD against poor initialisation also prompted Börsig, Brandes, and Pasztor (2020) to replace PivotMDS as the initialisation step with SGD itself, with majorization kept as the refinement stage. This combination resulted in the best performance of all, signifying that majorization is better at reaching the very bottom of local minima than SGD, whereas SGD is better at escaping poor overall minima. This is likely due to the fact that the recommended annealing schedule from Section 2.2.1 does not mathematically converge in expectation. This combination of SGD plus majorization was chosen for the layout implementation in EcoBuilder, outlined in Section 5.4.1.

2.3 Sparse approximation

This section contains an application of SGD to the sparse stress model of Ortmann, Klimenta, and Brandes (2017). The model is made sparse by strategically omitting terms in the summation of the objective function, by choosing a set of vertices as pivots and only considering shortest paths from these pivots.

To approximate the full model well it is important to choose pivots that are well distributed over the graph, and so Ortmann, Klimenta, and Brandes (2017) presented an experimental evaluation of various methods for doing so. The implementation here uses *max/min random sp* to select pivots. This was chosen because it gives good performance (second-best of all methods evaluated in terms of average reduction in stress) and the only method that performed slightly better, *k-means sp*, is more complicated to implement and has a worse asymptotic complexity (Ortmann, Klimenta, and Brandes, 2017).

To further elaborate, ordinary (non-random) *max/min sp* starts by picking one pivot randomly and computing its shortest paths to all other vertices, with subsequent pivots chosen by picking the vertex with the maximum shortest path to any pivot chosen so far (de Silva and Tenenbaum, 2004). The random extension *max/min random sp* instead samples for subsequent pivots with a probability proportional to shortest paths to any pivot, rather than simply always picking the maximum. *k-means sp* is also built upon *max/min sp*, as it uses the calculated shortest paths as a high-dimensional embedding, exactly following the same first step as the layout method of Harel and Koren (2004). A standard k-means clustering algorithm is then applied to this embedding (Friedman, Hastie, and Tibshirani, 2001b), and pivots are chosen as the vertices closest the centroids of these clusters. This extra clustering step is what pushes up the resulting asymptotic complexity (Ortmann, Klimenta, and Brandes, 2017) which, in combination with

Algorithm 2: Sparse SGD

SparseSGD (G, h)**:**
  inputs : graph $G = (V, E)$, number of pivots h
  output: k-dimensional layout $\mathbf{X}$
  $P \leftarrow$ MaxMinRandomSP(G, h)
  $d_{pi} \leftarrow$ SparseShortestPaths(G, P)
  $w'_{pi} \leftarrow 0$
  foreach $\{p, i\} \in (P \times V) : p \notin N(i)$**:**
    $s \leftarrow |\{j \in R(p) : d_{pj} \leq d_{pi}/2\}|$
    $w'_{ip} \leftarrow s\, w_{ip}$
  foreach $\{i, j\} \in E$**:**
    $w'_{ij} \leftarrow w'_{ji} \leftarrow w_{ij}$
  $\mathbf{X} \leftarrow$ Rand$(|V|, k)$
  for η in AnnealingSchedule()**:**
    foreach $\{i, j\} \in E \cup (P \times V)$ in random order**:**
      $\mu_i \leftarrow Min(w'_{ij}\eta\,,\, 1)$
      $\mu_j \leftarrow Min(w'_{ji}\eta\,,\, 1)$
      $\mathbf{r} \leftarrow \frac{||\mathbf{X}_i - \mathbf{X}_j|| - d_{ij}}{2} \frac{\mathbf{X}_i - \mathbf{X}_j}{||\mathbf{X}_i - \mathbf{X}_j||}$
      $\mathbf{X}_i \leftarrow \mathbf{X}_i - \mu_i \mathbf{r}$
      $\mathbf{X}_j \leftarrow \mathbf{X}_j + \mu_j \mathbf{r}$

Figure 2.14: Pseudocode for performing SGD on the sparse stress approximation described in Section 2.3. Note that all $R(p)$ and w'_{ip} can be constructed over the course of shortest path calculations without increasing the asymptotic complexity (Ortmann, Klimenta, and Brandes, 2017) by using a multi-source shortest paths algorithm.

the extra implementation work required, was why it was not chosen here.

These pivots $p \in P$ are then each assigned a region $R(p)$, which is the set of vertices closer to that pivot than any other. The relevant weights w_{ip} are then adapted depending on the composition of the region, resulting in a new second half of Equation (2.17) with fewer terms

$$\text{stress}(\mathbf{X}) = \sum_{\{i,j\} \in E} w_{ij}\sigma_{ij} + \sum_{i \in V} \sum_{p \in P \setminus N(i)} w'_{ip}\sigma_{ij} \tag{2.28}$$

where $\sigma_{ij} = (||\mathbf{X}_i - \mathbf{X}_j|| - d_{ij})^2$ and $N(i)$ are the neighbors of i to prevent over-

lap with any edges in the first summation. The adapted weight w'_{ip} is then set to $s_{ip}w_{ip}$, where s_{ip} is the number of vertices in $R(p)$ at least as close to p as to i, defined as

$$s_{ip} = |\{j \in R(p) : d_{jp} \leq d_{ip}/2\}|. \tag{2.29}$$

The reason the weight on vertex i is increased like this is because its contribution acts as an approximation for the stress to all vertices in $R(p)$, and (2.29) is required to prevent the weight on closer vertices from being overestimated. It is important to note that if both vertices p and q are pivots then w'_{pq} may not equal w'_{qp}. Importantly, if only p is a pivot then $w'_{pq} = 0$ as q should not contribute to the position of p.

This leaves far fewer terms in the summation, and so the complexity per iteration for SGD is now $\mathcal{O}(|E| + |P||V|)$, as any terms that have been cut are simply ignored. In the context of majorization this new model must be used with the local method described at the end of Section 2.1.3, Equation (2.12), resulting in the same overall complexity.

Resulting layouts are presented in Figures 2.15, where the graph `EVA` is the least well approximated by the sparse model, likely due to its low diameter and high degree distribution. `3elt` is very well approximated by the model itself, but the performance of majorization declines as the number of pivots exceeds ~100, likely due to the increased number of terms in the summation that introduce more local minima to jump over.

Note that the results here again do not use PivotMDS (Brandes and Pich, 2007) to initialize as done by Ortmann, Klimenta, and Brandes (2017), and instead initialize randomly within a 1×1 square as before. A gallery of larger graphs is presented in and 2.16, and pseudocode for the entire algorithm can be seen in Algorithm 2, Figure 2.14.

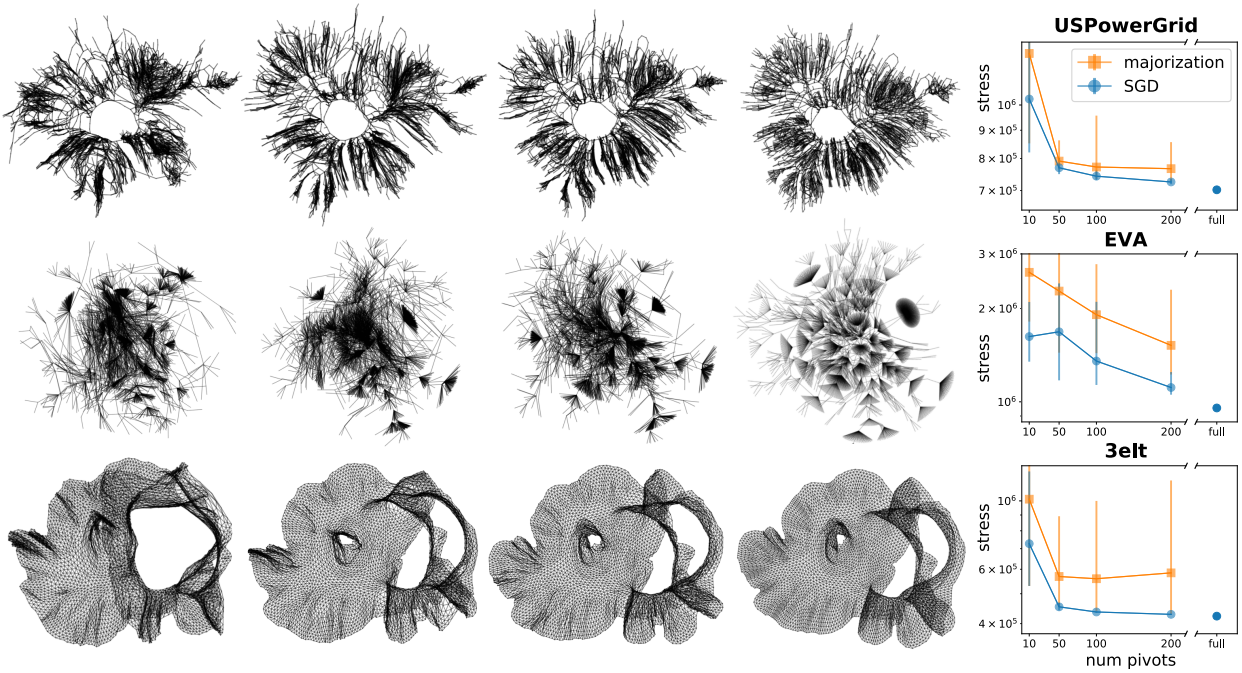


Figure 2.15: Examples of sparse SGD on the graphs `USPowerGrid`, `EVA`, and `3elt`. From left to right: best layouts using 10 pivots, 50, 200, full stress, and plots showing stress over 25 runs for each number of pivots. The 15 iteration annealing schedule from Section 2.2.1 is used for SGD, and majorization is allowed to run for 100 iterations.

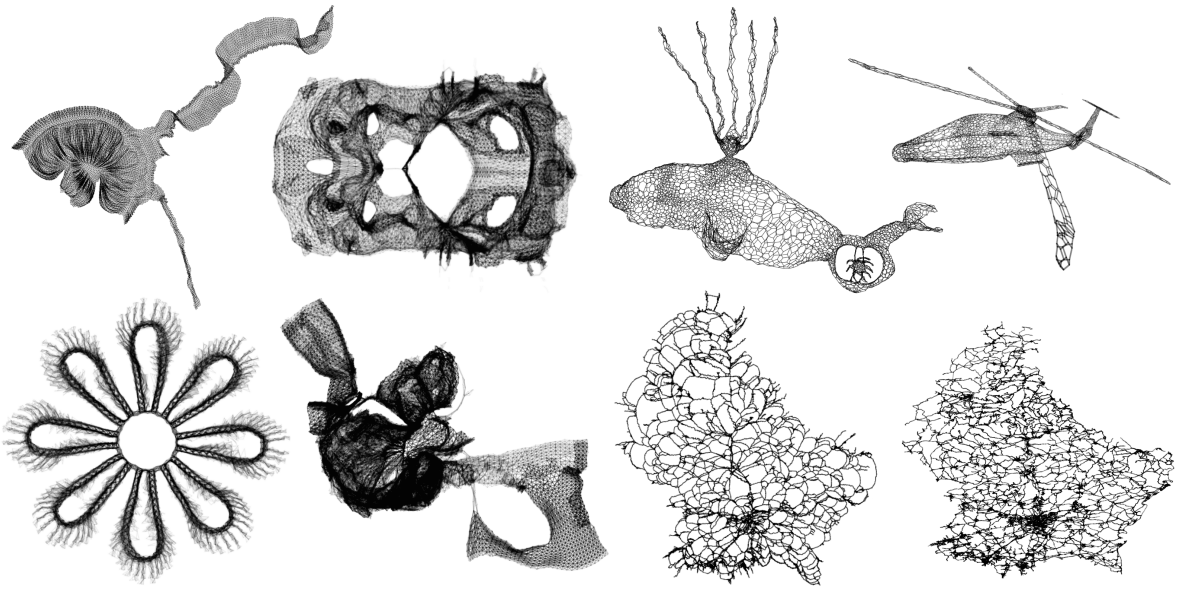


Figure 2.16: Some larger graphs, each approximated with 200 pivots. From left to right, top row then bottom: `pesa` (11,738 vertices), `bcsstk31` (35,588), `commanche_dual` (7,920) and its original (3D) layout, `finance256` (37,376), `bcsstk32` (44,609), `luxembourg_osm` (114,599) and its original layout. The right-hand graphs have edge weights calculated from distances in their original layouts.

2.4 Applications

A benefit of force-directed algorithms is the possibility of making ad hoc changes to the optimisation process, to change the visualisation in a domain-specific manner. For example, nails can be used to hold a node in place (Mi et al., 2016), magnetic fields can be used to align nodes in various ways (Sugiyama and Misue, 1994), and complex attractive and repulsive forces can be used to highlight clusters (Suh et al., 2019).

Some of the properties of SGD, in particular the fact that each edge is considered separately along with the ability to consistently avoid local minima well, make SGD well suited to such additional modifications. This section will describe some recipes for examples of this, each applied to various real-world graphs in order to show the merits of their use. Note that these applications are also possible with majorization, but can require more drastic modifications in order to apply them successfully.

2.4.1 Radial layout

It is often the case that a user will want to examine specific vertices in a graph, especially in an interactive setting. It is therefore important to be able to emphasize distances involving certain vertices. Brandes and Pich (2011) presented a general method of doing this in the context of majorization, by interpolating between two stress summations representing general and constrained weights separately.

For SGD, emphasizing specific distances is as simple as weighting the corresponding constraints more heavily. For example to focus on vertex 3, the relevant weights are set to infinity

$$w_{ij} \leftarrow \infty \quad \text{if } i = 3 \text{ or } j = 3. \tag{2.30}$$

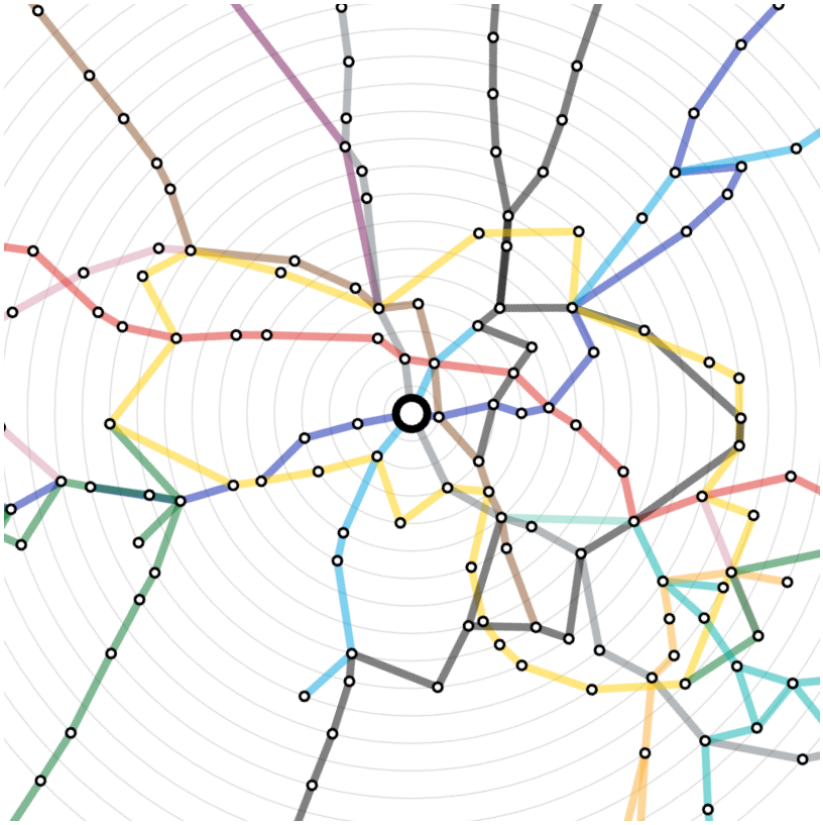


Figure 2.17: The London Underground map with a focus on Green Park, created using the method described in Section 2.4.1. The distances are based on travel time rather than real world distance.

This causes only the remaining constraints to decay, but the system still converges in this case as there are no conflicts between the ones emphasized. Exploding gradients are still prevented by the capping of the step size in Equation (2.21). An important remaining step is that the step size must be allowed to decay for extra iterations, in order to allow the infinitely weighted terms to essentially become the only remaining contribution at convergence. An example of this method in action can be seen in Figure 2.17.[4]

Setting weights to infinity when using majorization results in the algorithm becoming instantly very stuck, which is why the more complicated interpolation (Brandes and Pich, 2011) is necessary.

[4]Data collected from www.github.com/petertrotman/london-underground-travel-time-map, accessed 13/8/20.

This idea of editing individual constraints can also be used to pin nodes in place. This could be done by not allowing pinned nodes to be moved when satisfying the constraint as in Figure 2.4. Additionally if many nodes are pinned, then any constraints between pinned nodes can be left out of the objective function entirely, reducing the amount of computation time required.

2.4.2 General multidimensional scaling

Stress as an energy function originated as a general visualisation algorithm for embedding high-dimensional data, and using SGD to optimize it does not change its ability to perform this original function. Here this will be shown by applying it on a distance measure other than the graph-theoretic shortest path, in order to show its versatility. The measure that will be used is the Jaccard index, which can be defined as a distance by

$$d_{ij} = 1 - \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|} \tag{2.31}$$

where $N(i)$ is the set of neighbors of vertex i. This is effectively a measure of how similar the neighbour sets of vertices i and j are to each other. This measure will be applied on top of an existing graph layout, but since the two spatial dimensions are already being used, the three-dimensional space of *colour* will be used instead.

Since color is simply a linear mix of red, green, and blue (RGB), it can be used as a 3D space in which Euclidean distances can be embedded, where each color corresponds to a separate axis. Note that using the raw RGB space to embed colour is not generally recommended due to its nonuniformity causing unwanted visual artifacts (Rogowitz and Treinish, 1998). Ideally a perceptually uniform colour space would be used such as CIELAB (Smart, Wu, and Szafir, 2019), but in this small illustrative example it was not vital.

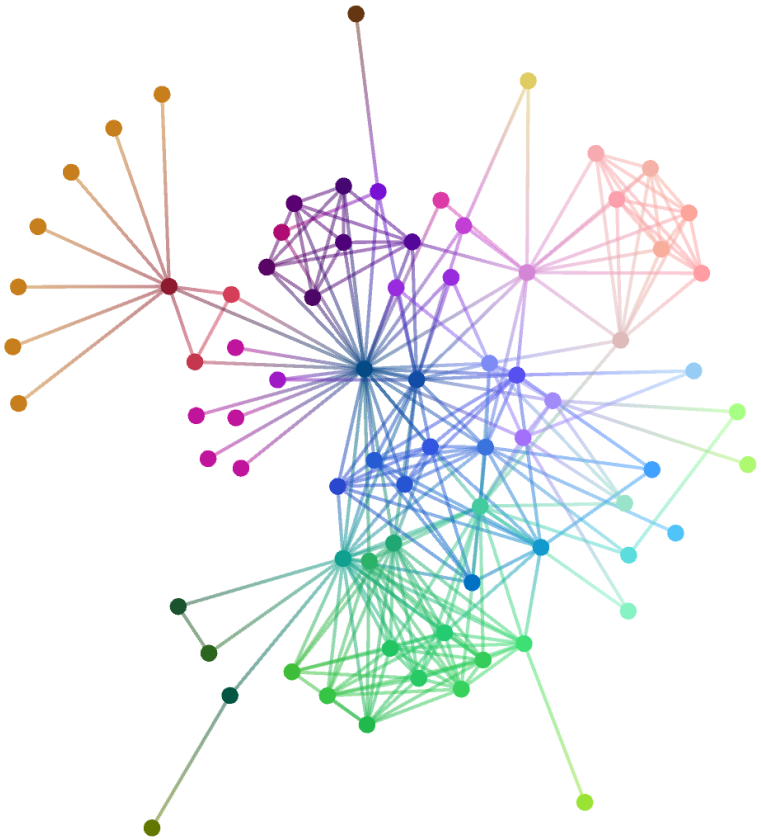


Figure 2.18: Co-appearances of characters in Les Misérables by Victor Hugo (Knuth, 1993). Groups of similarly colored vertices indicate clusters based on Jaccard dissimilarity as in Equation (2.31).

Since the Jaccard distance is bounded between $0 \leq d_{ij} \leq 1$, embedded distances fit perfectly within the similarly bounded axes of color. This means that vertices not only have coordinates within normal Euclidean space, but also within RGB space. An illustration of this can be seen in Figure 2.18, where vertices are coloured according to their (secondary) embedding in RGB space. Note that in this case w_{ij} is simply set to one.

This process can help to reveal groupings, but can also produce ambiguity when applied to larger graphs due to the lack of distinct color combinations, again a problem caused by a lack of output dimensions. One possibility in this case would be to use an interactive form of visualization in which the user selects a smaller group of vertices at a time, and the algorithm embeds only their selection in an

RGB space, by considering dissimilarities only between selected vertices.

2.5 Discussion

One of the major reasons why previous force-directed algorithms (Eades, 1984; Kamada and Kawai, 1989; Fruchterman and Reingold, 1991; Frick, Ludwig, and Mehldau, 1995), have become popular is how simple and intuitive the concept is. The idea of a physical system pushing and pulling vertices, modeled as sets of springs and electrical forces, makes them easy to understand and quick to implement for practical use.

The geometric interpretation of the SGD algorithm presented here shares these qualities, as the concept of moving pairs of vertices one by one towards an ideal distance is just as simple. In fact the stress energy function in Equation (2.8) is commonly known as the *spring model* (Kamada and Kawai, 1989; Hu, 2005), and so the physical analogy of SGD to decompressing one spring at a time very naturally fits this intuition. The implementation also requires no linear equation solver, and there is no need to consider smart initialization, which can often be just as complex a task (Brandes and Pich, 2008). Considering only a single pair of vertices at a time also makes further constrained layouts easy to implement, and allows an appropriate sparse approximation to grant scalability up to large graphs.

But perhaps the most important benefit of SGD is its consistency regardless of initialization, despite being non-deterministic due to the shuffling of the order of terms. This claim was further supported by Börsig, Brandes, and Pasztor (2020) in their replication study of the work done here, as outlined in Section 2.2.4. By contrast, the plots in Section 2.2.3 clearly show how vastly the results from majorization can differ depending on initialization, especially when restricted to a

limited number of iterations. This reliability of SGD can be crucial for real-time applications with fixed limits on computation time, such as within an interactive visualization.

However there are still situations where SGD can struggle with local minima, such as `dwt_2680` which is susceptible to twisting in the middle. This can be seen in Figure 2.13 where a twisted layout is purposefully included to illustrate this pitfall. A potential solution to this is overshooting, or in other words allowing values of $1 < \mu < 2$ in Equation (2.21). This greatly reduces the chance of a twist, but results in poorer local minima in most other cases and can also bring back the problem of divergence, so is a potential avenue for future work, perhaps to be used in conjunction with an adaptive annealing schedule to further optimize performance depending on the input data.

This will be used in Chapter 5, Section 5.4.1 for a real-world application, where it will be shown that allowing a slightly higher maximum step size is in fact necessary for another domain-specific application, on top of the ones shown in Section 2.4.

Interaction could also allow the user to manually adjust the step size η from Equation (2.20), allowing them to 'shake' the graph out of local minima themselves. The step size annealing from Section 2.2.1 is the most ad hoc and data-dependent component of SGD, so handing control over to the user could help, especially in dynamical situations where the structure of the graph changes with time.

Additionally, if frame rate becomes an issue in an interactive setting, the application does not have to wait until the end of an entire iteration before rendering an updated layout, because vertices are continually being moved at each satisfaction step. This would keep the application smooth and responsive, whilst still giving an indication of the algorithm's progress.

The application of gradient-based methods to functions such as stress has also re-

cently been explored by Ahmed et al. (2020), who further showed their flexibility by supplementing the objective function with objectives such as angular resolution or number of crossings. Their work especially shows the potential benefits of taking inspiration from the rapidly growing machine learning literature through their usage of tools such as `tensorflow.js` (Smilkov et al., 2019).

Conclusion

This chapter has presented a modified version of stochastic gradient descent (SGD) to minimize stress as defined by Equation (2.8). An investigation comparing the method to majorization shows consistently faster convergence to lower stress levels, and the fact that only a single pair of vertices is considered at a time makes it well suited for variants such as domain-specific modifications or the pivot-based approximation of Ortmann, Klimenta, and Brandes (2017). This improved performance — combined with a simplicity that forgoes an equation solver or smart initialization — makes SGD a strong candidate for general graph layout applications.

All code used for timing experiments, along with a fast C++ implementation with a Python wrapper, is open source and available at `www.github.com/jxz12/s_gd2`.

Chapter 3

Hierarchical edge bundling

The previous chapter was an exploration of how to position the vertices of a graph, and the natural question to then ask is how to deal with the only remaining component: the edges. However this question is seemingly redundant at first, as the obvious answer is to simply draw straight lines between adjacent nodes. While this is by no means a poor choice, and is exactly how all node-link diagrams have been drawn thus far in this thesis, this chapter and the next will explore the possibility of drawing links using curves instead.

3.1 Background

The curving of links in the context of a node-link diagram is known as *edge bundling*. It is a technique that has been developed because many networks, when processed through a standard force-directed layout, result in a seemingly random layout with no discernable structure. See Figure 3.7 for examples of this. The similarity of such layouts to tangled collections of hair has led to them being colloquially termed *hairballs*.

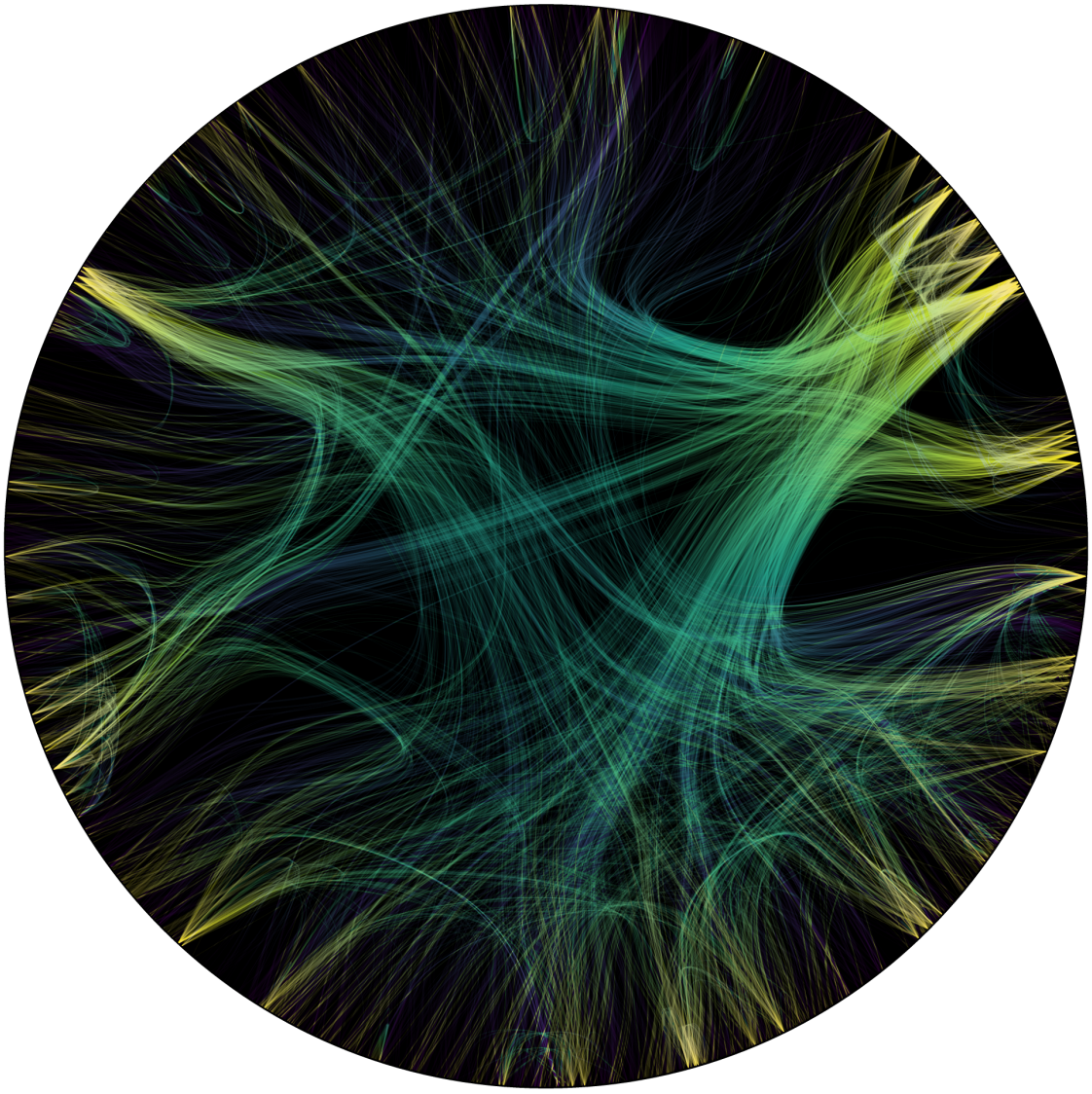


Figure 3.1: A visualisation of the Unity game engine, rendered using the Unity game engine. Edges are bundled using hierarchical edge bundling as described in Section 3.1.1. The graph has a total of 5,025 vertices and 20,177 edges, reduced to 8,380 included in the final render to reduce clutter. The colour applied to each edge highlights the class being referenced by transitioning from light to dark, in order to emphasise the pattern of certain important nodes that many others depend on. For example, the thickest bundle in the top right comes from a folder called `Math/`, which is used in calculations all over the source code. The thick bundle coming from the top left is a folder called `IMGUI/`, which is used for debugging with developer tools. Using `MAX_EXT` blending as suggested by Holten (2006) allows links flowing in opposite directions to be salient, and anti-aliasing is crucial for smoothing the output. Source code for generating this figure can be found at `www.github.com/jxz12/HierarchicalEdgeBundles`.

Unfortunately this is not an easily solved problem, because the *curse of dimensionality* (Friedman, Hastie, and Tibshirani, 2001a) means that most of these networks simply cannot be accurately represented in two dimensions, and the likelihood of this problem only rises as the size of any network increases. This can happen even if there exists a clear underlying structure to these networks, lying beyond the reach of the standard layout algorithm. Edge bundling attempts to alleviate this issue by introducing a trade-off: the ability to follow individual links is sacrificed for better representation of global structure, by allowing links to overlap. This is analogous to organising the wires in a computer system by tying groups of wires together that share similar endpoints.

The literature is rich with various different methods for performing this bundling, dating all the way back to the 1800s with the flow maps of Minard (1862) or Sankey diagrams (Sankey, 1896). More modern algorithms for performing bundling automatically include multilevel agglomerative edge bundling by Gansner, Hu, North, and Scheidegger (2011) which greedily merges pairs of links at a time by selecting the pairs that minimise a cost function based on the amount of 'ink' used to draw links. A more complex cost function is used in metro-style bundling by Pupyrev et al. (2016), which is based on multiple criteria including ink, individual edge lengths and node separations. The winding roads method of Lambert, Bourqui, and Auber (2010) first draws a supplementary grid around the nodes in the graph, and routes edges through this grid such that they naturally bundle on shared grid coordinates.

The same premise behind force-directed node layout is also used in force-directed edge bundling by Holten and van Wijk (2009), which bundles links by defining forces between adjacent links instead of nodes. As shown previously in Section 2.1.1, force-directed methods are in fact gradient descent optimisation methods where the gradient is defined before the cost function itself, and this edge bundling tech-

nique is no different. A similar method by Cui, Zhou, et al. (2008) also uses geometry to group edges, but instead discretely clusters them first.

Another effective approach is *kernel density estimation* by Hurter, Ersoy, and Telea (2012), who iteratively apply a convolutional filter over a density map of links in an already rendered diagram. This method belongs to the subfield of image-based bundling methods (Lhuillier, Hurter, and Telea, 2017; Telea, 2018). A diverse gallery of edge bundling algorithms applied to the same dataset can be seen in the review of Lhuillier, Hurter, and Telea (2017).

However all of the aforementioned methods share a key similarity: they apply bundling upon the assumption that node positions are predetermined and will not be moved. This is perfectly fine and even desirable in many common use cases where nodes have a predefined location, such as geographical maps. However when there is no predefined positioning, the usual methods such as force-directed layouts were never designed to lend themselves towards bundling. A layout that can be bundled must place similar links in parallel, and this is not guaranteed or even desired for most force layouts. Maximising the angular resolution of links sharing nodes is sometimes even directly optimised as part of the cost function (Argyriou, Bekos, and Symvonis, 2010) and so common layout methods usually do not help, and may even worsen, the bundling quality of their visualisations.

3.1.1 Hierarchical bundling

Avoiding this issue, of common node layouts not lending themselves towards bundling, is why one of the most powerful bundling techniques is a method known as *hierarchical edge bundling*, published by Holten (2006)[1]. This is because the layout of vertices is directly informed by the structure of the bundles. An exam-

[1] Its effectiveness was recognised by a *Test of Time* award at the IEEE VIS conference in 2016.

ple of this in action can be seen in Figure 3.1, which illustrates the method applied to the source code of the Unity game engine. The graph being visualised is a call graph, where each class is a vertex, and is connected by an edge to any other class it depends on.

The trick to the bundling comes from the fact that there is extra metadata available: the hierarchy of folders and source files that hold the code. This hierarchy, not the call graph itself, is first layed out as a tree. Then, since this tree includes extra nodes to represent the folders and source files, these extra nodes are erased from the final visualisation. However their coordinates are instead used as an auxiliary routing graph (ARG) to route the original edges through. More precisely, this process of routing involves taking every edge in the original call graph (the vertices of which map to leaves of the hierarchical tree) and rendering each one using a spline curve, whose control points consist of the path from leaf to leaf through the ARG. There is only one such path through the ARG because it is a tree, although this may not necessarily be the case; this possibility of a non-tree-shaped ARG will be studied in the following chapter, Section 4.2.3. The precise definition of the spline curve is also important, and will be further elaborated upon in Section 4.2.5, but for now the important thing to note is that this produces bundling behaviour because vertices close to each other in the hierarchy will share control points for their splines, thus curving their rendered links towards each other. An example of a hierarchical tree with its resulting bundles side by side can be seen later in Figure 3.3.

This technique is so effective because the structure of the bundles is supervised by human judgement. To continue within the call graph example, the hierarchy of folders and source files is literally designed by the programmers for organisational purposes, and so it is very likely to admit some utility for also organising, say, a visualisation. Additionally, not only is the curvature links influenced by the

ARG, but so is the position of the nodes. Since each node is a leaf in the tree-shaped ARG, their positions around the circle are also influenced by the topology of the tree. This idea of using an auxiliary hierarchy to inform bundling is the basis of the work in this chapter, except that it will instead investigate the more common situation of when this metadata is not included with the original data. It must therefore must be generated by the algorithm itself as a preprocessing step.

3.1.2 Clustering

To generate a hierarchy, what is needed is a way of grouping similar datapoints together. This is known as *clustering*, and is a vast field of study, not least as one of the primary objectives of the fast-growing discipline of machine learning. The subfield of clustering just within the context of networks is large in its own right, due to its utility in common datasets such as protein–protein interaction or social networks.

There exist a wide variety of powerful and creative methods that have been developed to perform clustering on networks. A widely used method is to define an equation that can be used to numerically measure the quality of a given clustering, and to attempt to maximise this quality. This is reminiscent of the optimisation-based methods described in Section 2.1.1. A popular version of this is known as *modularity*, defined as

$$Q = \frac{1}{|E|} \sum_{i,j} \left(\mathbf{A}_{ij} - \frac{|N(i)||N(j)|}{|E|} \right) \mathbf{C}_{ij} \tag{3.1}$$

where $\mathbf{A}$ is the adjacency matrix where $\mathbf{A}_{ij}$ equals 1 if vertices i and j are connected by an edge, and $\mathbf{C}_{ij}$ equals 1 if vertices i and j are in the same cluster and 0 otherwise. The value of Q lies between $-1/2$ and 1 (Brandes, Delling, et al., 2007), and it

can be intuitively understood as the fraction of the edges that fall within the given clusters minus the expected fraction if edges were distributed at random.

This was introduced by Newman and Girvan (2004) to evaluate a separate clustering algorithm, and later directly optimised by Newman (2006b) with two methods that both initialise the network as one big cluster that is recursively split into half until no further gain in modularity is possible. The first finds the split by calculating eigenvectors of the *modularity matrix*, and another that finds the split by repeatedly moving vertices between the two sides of the split until modularity is maximised, akin to a hill-climbing method. Optimising modularity in general is $\mathcal{NP}$-complete (Brandes, Delling, et al., 2007), but its intuitive interpretation has led to various other authors to develop methods to optimise it. The Louvain algorithm (given its name from the authors coming from the University of Louvain, Belgium) (Blondel et al., 2008) and its recent improvement to the Leiden algorithm (from Leiden University, Netherlands) (Traag, Waltman, and van Eck, 2019) both start by placing every vertex in a singleton cluster, followed by successively merging clusters whilst also moving vertices between clusters, also until modularity cannot be improved further.

The behaviour of random walks along the graph has also been used to great effect for clustering. For example, the *markov cluster* algorithm of Enright, van Dongen, and Ouzounis (2002) simulates a random walk process through a series of matrix multiplications on the markov chain of the graph. The *infomap* algorithm of Rosvall and Bergstrom (2008) optimises a cost function known as the map equation, which applies information-theoretic concepts such as entropy to the behaviour of random walkers. It is optimised through a combination of greedy search and simulated annealing. Other types of algorithms include statistical inference methods such as Bayesian inference (Hastings, 2006) and block-modelling (Reichardt and White, 2007). Another perspective on clustering includes grouping together

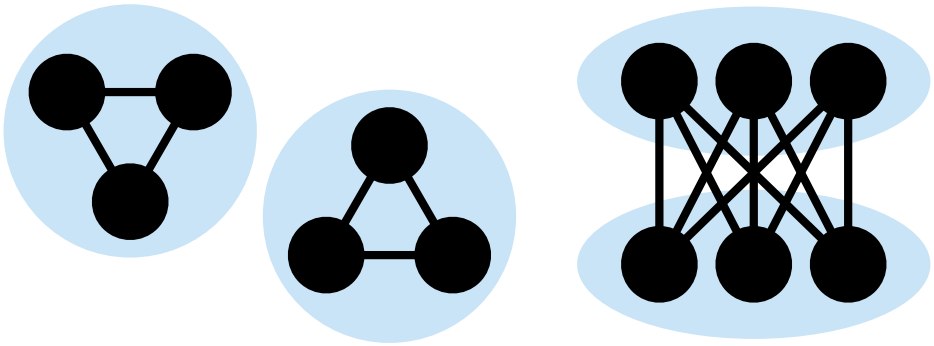


Figure 3.2: An example of different types of clusters. On the left is a situation with edges within clusters, and on the right with edges between clusters.

edges instead of vertices, which means that a single vertex may belong to many overlapping clusters, an idea explored by Evans and Lambiotte (2009).

The growing number of graph-shaped datasets being used in machine learning has also led to a number of supervised and semi-supervised methods for clustering and general dimensionality reduction, such as *DeepWalk* (Perozzi, Al-Rfou, and Skiena, 2014) which treats each vertex as a 'word' to generate 'sentences' using random walks across vertices. These sentences are then taken and applied to techniques originally designed for language processing. Attempts to transfer the success of convolutional neural networks have also been generalised to apply to graphs of any topology (Kipf and Welling, 2016; LeCun et al., 2017), as well as many other architectures depending on the task at hand (Battaglia et al., 2018; Wu et al., 2020).

One reason why there is such a variety of clustering algorithms is due in part to it being an undefined problem. A simple example of two equally valid but opposing definitions for a cluster would be to compare a clique vs. a biclique. A definition that attempts to keep edges within clusters, such as Equation 3.1, perfectly captures cliques, but fails for bicliques, as none of the edges are within the clusters, instead all falling between them. It is therefore important to keep in mind that clustering algorithms are attempts to quantify common characteristics of real-world

data, and there is generally no 'right answer' to the question of which one should be chosen; it depends on both the specific objective and dataset being examined. Nevertheless, they have been used to great effect in many applications (Fortunato and Hric, 2016), and so confidence can be placed in their utility as a tool to help understand patterns in data.

The work in this chapter will focus on *hierarchical clustering*, the family of clustering algorithms where the output is a *dendrogram:* a binary tree used to describe a hierarchical structure. This can either be done from the bottom up, i.e. each node starts in its own cluster and pairs of clusters are progressively merged until only one remains, or top down, i.e every node starts in the same cluster that is recursively split into two halves. The former is known as *agglomerative* clustering, and the latter as *divisive* clustering. The output fits well here, since the goal is to construct a hierarchical edge bundling visualisation like the one in Figure 3.1, but without a prior hierarchy included with the data; the dendrogram produced from such a clustering algorithm can be used as this missing hierarchy.

This idea of using the output dendrogram of a clustering algorithm to produce hierarchical edge bundling has been previously explored by Jia, Garland, and Hart (2011) who produce a hierarchy using the algorithm of Girvan and Newman (2002). This method removes one edge from the network at a time, specifically the one with the most *betweenness centrality* i.e. the edge traversed the most often when mapping the shortest paths between all pairs of vertices. This naturally splits clusters when an edge is removed that is the final bridge between two clusters.

Unfortunately Jia et al. did not perform any quantitative analysis to evaluate the performance of their clustering algorithm, and only present results when using one clustering method. The contribution in this chapter extends their work by evaluating the performance of a number of hierarchical clustering methods, against a benchmark of graphs where the ground truth cluster configuration is known.

3.1.3 Hierarchical clustering

To assess the quality of the divisive Girvan-Newman (GN) algorithm used by Jia, Garland, and Hart (2011), its performance will be compared with *agglomerative dissimilarity clustering*. This comparison will be done via experiment in Section 3.2, and the rest of this background section will outline the different methods it will be compared to.

Dissimilarity clustering is a family of algorithms where the input is a matrix of *dissimilarities* between individual datapoints, in this case the vertices of a graph, and the output is a dendrogram as required. Similarly to the stress-based layout explored in the previous chapter, it is a more general algorithm that can be applied to any numerical dataset and not just graphs, unlike the methods previously outlined in Section 3.1.2. It is also closely related to the class of multidimensional scaling methods that stress minimisation belongs to, as both procedures take a matrix of dissimilarities as input.

Not only does this fit thematically with the topics explored in previous chapters, but a key benefit to using dissimilarity clustering is that the output dendrogram contains extra information to describe the 'quality' of the merge at that point. This information is vital as it can be used to 'prune' the hierarchical tree in order to make the resulting edge bundles more salient. This will be elaborated upon in Section 3.2.2, and an example of the method pipeline from start to finish is illustrated in Figure 3.3. Two other questions must first be answered before one can proceed with performing this clustering: what exactly are the dissimilarities to be used as input, and how will the algorithm proceed to process these dissimilarities to merge clusters? There are many choices for answering these questions, and outlining these will cover the remainder of the background for this chapter.

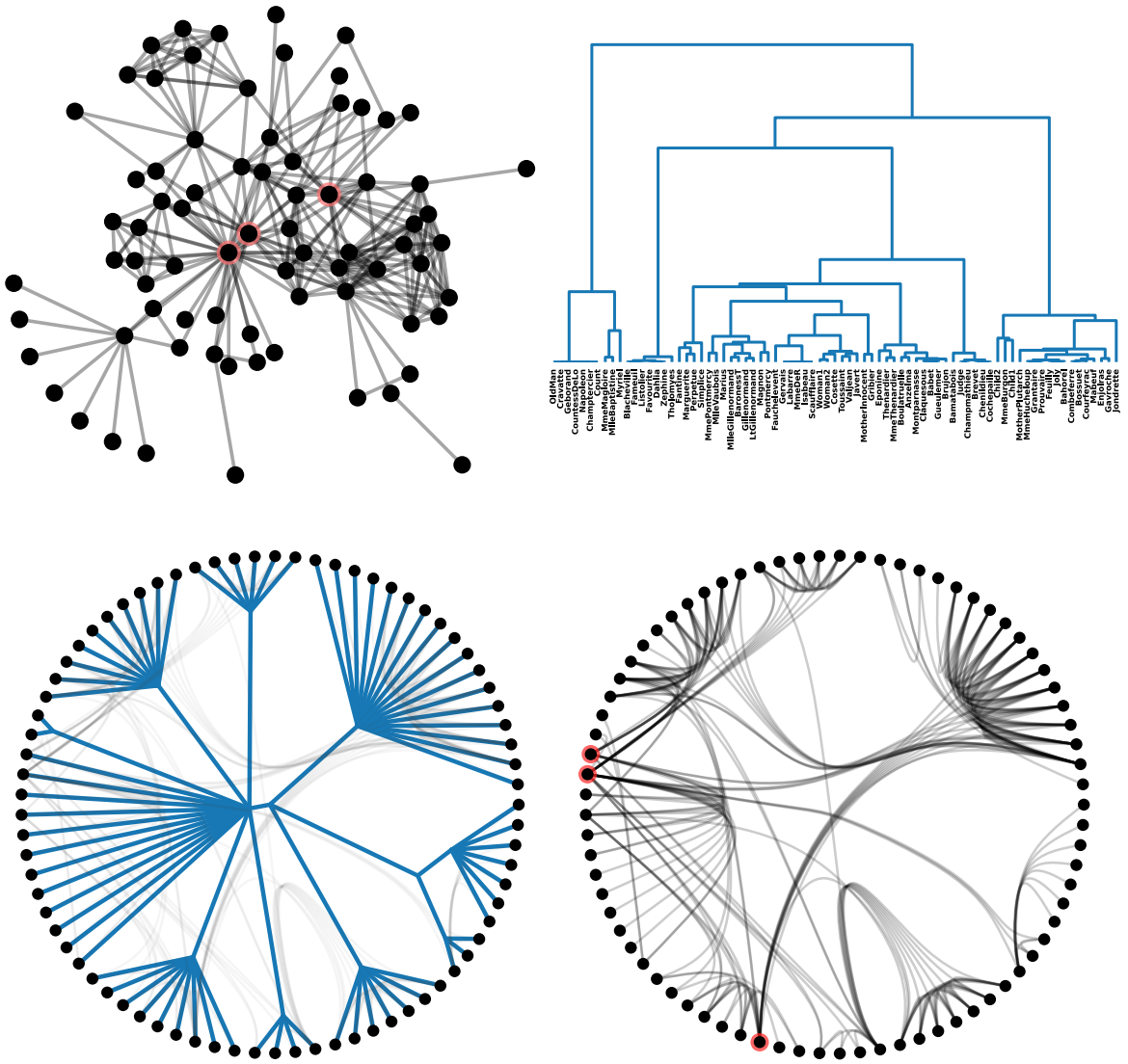


Figure 3.3: An example of the hierarchical edge bundling pipeline explored here, on the graph `lesmis` which describes co-appearances of characters in the musical les Misérables (Knuth, 1993). The top-left is a layout from stress optimisation; top-right is a dendrogram computed from agglomerative dissimilarity clustering (see Section 3.1.3); bottom-left is the tidied up hierarchical tree after pruning (see Section 3.2.2); bottom-right is the final hierarchical edge bundling diagram when original edges are rendered using splines (see Section 3.1.1). The characters Jean Valjean and Javert are the main protagonist and antagonist respectively, and they are placed next to each other for both the stress and bundled layouts. Those two are highlighted, as well as a third main character, Marius, whose role in the network is much clearer in the bundled layout.

Greedy agglomeration

General hierarchical clustering based on dissimilarities can be done in both an agglomerative and divisive way, but the work here will focus on agglomerative methods. Agglomerative methods are more popular among practitioners (Roux, 2018) due largely to the divisive version having $\mathcal{O}(2^{|V|})$ possible splits for a cluster of size $|V|$, as each datapoint can be placed in either one half or the other. This results in an exponential runtime complexity per split, whereas the agglomerative method has only to chose from pairs of datapoints, giving an $\mathcal{O}(|V|^2)$ complexity per merge.

There do exist heuristics to speed up the divisive method to bring the complexity down to polynomial, such as using algorithms like k-means to decide the split instead (Lamrous and Taileb, 2006). However, since there already exists a graph-specific divisive method in the GN algorithm to compare to, this chapter will focus on comparing it to its agglomerative counterparts.

The agglomerative process is as follows:

1. Calculate the dissimilarities between all pairs of datapoints.
2. Place each datapoint into its own singleton cluster.
3. Iterate over each pair of clusters, and merge the closest pair into a single cluster.
4. Recalculate the dissimilarities between the newly merged cluster and the remaining unmerged clusters.
5. Repeat steps 3 and 4 until only one supercluster remains.

On the surface this is a simple procedure, but it is not immediately clear how to calculate dissimilarities between datapoints or between clusters in steps 1 or 4.

There are an abundance of choices for what to use for step 1, which is the same first step as used in stress-minimisation from Section 2.1.2. There, the graph-theoretic shortest-path was used as the dissimilarity, but there exist a wide variety of other measures that can be used that result in higher quality results. This will be shown in the experimental comparison in Section 3.2 and discussed later in this section. Here the assumption that a dissimilarity is already defined between datapoints will first be taken, to begin by outlining how they are recalculated when clusters are merged.

There are also many choices for recalculating dissimilarities between clusters, and each choice can be thought of as a different heuristic for the greedy choice of which pair of clusters to merge together. The most commonly used methods fall into three categories: using one member of a cluster as a representative, averaging over all cluster members, and measuring the variance of members within a cluster (Roux, 2018). Choosing a single representative member is the most straightforward; when calculating the dissimilarity between two clusters, one can take the best-case scenario by choosing the closest pair of inter-cluster points, or the worst-case by choosing the farthest pair. These are known as single- and complete-linkage algorithms, respectively (Sibson, 1973; Defays, 1977). One can also choose to take an average over the members of both clusters. This can be done by taking the mean dissimilarity between all inter-cluster pairs, and and is known as UPGMA (Unweighted Pair Group Method with Arithmetic mean). Its 'weighted' counterpart, known as WPGMA, is more efficient, as when a cluster is merged it 'forgets' that it was made up of other elements and becomes treated as a single datapoint itself (Sneath and Sokal, 1973). The reason it is called weighted is, perhaps confusingly, because the original datapoints 'lose weight' by being merged into a cluster, whereas in the unweighted version each original datapoint still contributes the same amount to each merging decision.

The third category is known as Ward's method (Ward Jr, 1963), and is based on minimising variance within clusters. The dissimilarity between clusters in this case is the increase in total within-cluster variance if they were to be merged. This is the same objective function that k-means clustering attempts to minimise (Friedman, Hastie, and Tibshirani, 2001b).

Note that since each merge is based on the dissimilarity between clusters, this extra information that can be stored in the resulting dendrogram as the height of the branch. This is a direct benefit of using dissimilarity clustering, and is usually used to decide where to cut the dendrogram if flat clusters are needed. It will be leveraged to simplify the resulting dendrogram in Section 3.2.2. However this is also a source of trouble for Ward's method specifically, because a monotonically increasing branch height is only guaranteed if the input is a *distance*, and not just a *dissimilarity*. A distance must strictly satisfy the conditions

$$\begin{aligned} d_{ij} &\geq 0 \quad \text{and} \quad d_{ij} = 0 \Leftrightarrow i = j \\ d_{ij} &= d_{ji} \\ d_{ij} &\leq d_{ik} + d_{ky} \end{aligned} \tag{3.2}$$

where the third condition is known as the triangle inequality, and is a requirement for the points to have a corresponding embedding in a metric space, such as Euclidean. A dissimilarity is only required to satisfy the first two conditions. Since Ward's method is based on variance, a concept that only makes geometric sense in a metric space, using a dissimilarity may result in dendrograms with negative branch lengths, making it impossible to cut the tree and form flat clusters.

The results in Section 3.2 will include both distances and dissimilarities, both applied to Ward's method and the others reviewed in this section. In practice, any reasonable dissimilarity does not often result in negative branch lengths with Ward's method, even if it does not satisfy the criteria of a distance. None of the

results presented on the benchmark used here have this problem, but it is important to note the possibility of it happening in practice.

The final thing to discuss before moving onto how to choose a dissimilarity is the computational complexity on recalculating dissimilarities between clusters when they are merged. Remarkably, all of the above methods can be recalculated based on the Lance-Williams equation

$$d_{\{i\cup j\}k} = \alpha_i d_{ik} + \alpha_j d_{jk} + \beta d_{ij} + \gamma |d_{ik} - d_{jk}| \tag{3.3}$$

where $d_{\{i\cup j\}k}$ is the new distance between k and a newly merged cluster containing i and j, and the values of α, β, and γ can all be calculated in constant time. Recalculating the dissimilarity matrix after a merge therefore takes $\mathcal{O}(|V|)$ time. Searching through this matrix to find the best merge takes $\mathcal{O}(|V|^2)$ time, but this can be reduced to $\mathcal{O}(|V|\log(|V|)$ with a priority queue. Since forming a complete dendrogram requires $\mathcal{O}(|V|)$ merges, the overall complexity of greedy agglomerative clustering is $\mathcal{O}(|V|^2\log(|V|)$.

3.1.4 Random walks as dissimilarities

The first step before performing the agglomeration is to calculate the pairwise dissimilarities between individual datapoints. As previously noted, the shortest path distance was used in the context of optimising stress, but it be shown here that this is far from optimal in a clustering context. The reason for this that there is not enough *local* information to differentiate vertices successfully. For example, imagine a graph with two fully connected cliques, but with one edge connecting a single inter-cluster pair (one edge connecting the two clusters to the left of Figure 3.2). Since the shortest path between the pair is one hop, it means that the pair is considered just as similar to each other as to the other vertices in their respec-

tive cliques. Similarly, vertices in the same group within a biclique are not connected at all (right side of Figure 3.2) and so within-cluster pairs would be considered even less similar than inter-cluster pairs.

A commonly used distance to capture local detail is the Jaccard index

$$d_{ij} = 1 - \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|} \tag{3.4}$$

which can be interpreted as the proportion of neighbours shared between vertices i and j. However, this has the exact opposite problem to using shortest paths, as it fails to capture any *global* detail, because any pair of vertices more than two hops away can never share any neighbours, and so will always have a maximum dissimilarity of one. An ad hoc solution to this problem is to combine the two measures by simply multiplying them together, following Zheng, Pawar, and Goodman (2018). Note that despite shortest paths and Jaccard index both being distance metrics (Clarkson, 2006), it is not guaranteed the product of dissimilarities will also be.

A popular method of graph clustering that uses dissimilarities as input is the *cluster walktrap* method of Pons and Latapy (2006). They use *random walks* to first construct a high-dimensional embedding of each vertex. To construct the high-dimensional embedding, the *Markov chain* transition matrix of the graph is calculated, defined as

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A} \tag{3.5}$$

where $\mathbf{D}$ is a diagonal matrix of degrees, where the entries are the sum of the corresponding row in $\mathbf{A}$, which is the adjacency matrix of the graph that is not necessarily binary if edges have weights. $\mathbf{P}$ is essentially a normalised $\mathbf{A}$ where rows sum up to one.

However, this is not directly used as the high-dimensional embedding because of

the same problem as the Jaccard coefficient in Equation (3.4), that is it only captures local detail. This matrix is therefore multiplied by itself a number of times in order to simulate a longer random walk. However, multiplying too many times results in a matrix converging to a state only dependent on the in-degree of each vertex (Pons and Latapy, 2006), and so the length of the walk is left as an input parameter that should be set to an intermediate value. The distance is defined as

$$d_{ij} = \sqrt{\sum_k \frac{(\mathbf{P}^t_{ik} - \mathbf{P}^t_{jk})^2}{\mathbf{D}_{kk}}} \tag{3.6}$$

where $t > 0$ is an input parameter that determines how many times $\mathbf{P}$ should be multiplied by itself. Notice that this is very close to being simply the Euclidean distance between columns in $\mathbf{P}$, except with a factor consisting of $\mathbf{D}^{-1/2}_{kk}$. This was done presumably in order to undercompensate for high-degree vertices, although they state that their distance *"can also be seen as the L^2 distance between the two probability distributions"* (Pons and Latapy, 2006) without mentioning this factor. However a look into their provided source code confirms the use of the factor. Since this measure can also be written as the Euclidean distance between columns in the matrix $\mathbf{D}^{-1/2}\mathbf{P}^t$, it automatically also qualifies as a distance metric. Pons and Latapy (2006) did not study whether the inclusion of this factor performs better quantitatively, and the performance on the benchmark used in Section 3.2 is in fact always higher when this factor is left out. Equation (3.6) will therefore be used without including it, i.e. simply $\sqrt{\sum_k (\mathbf{P}^t_{ik} - \mathbf{P}^t_{jk})^2}$. This will be referred to as the *snapshot* embedding, as it captures the random walker state after a fixed number of steps t.

The algorithm then proceeds by applying agglomerative clustering using the Ward method, but with one difference: it only merges clusters connected by an edge. This reduces computation time and likely improves the modularity of the resulting clusters, but since the end goal is to produce bundles, edges between clusters

are acceptable as they will be bundled. The results here will therefore use the more general algorithm that is unaware of edges between clusters.

It is also not necessary to be restricted to having the embedding capture the random walker at a single walk length. Another type of embedding that will be used is the weighted average over all walk lengths. However this would also converge to the same matrix as an infinite length walk, by virtue of the Markov matrix converging to only depend on in-degrees as previously mentioned. In this case a *damping factor* is used, similar to the one used in PageRank (Page et al., 1999) and infomap clustering (Rosvall and Bergstrom, 2008) to simulate a random walker losing a proportion of its energy on each step and eventually coming to a stop. This is defined as

$$\mathbf{P}' = \sum_{k=0}^{\infty} t^k \mathbf{P}^k \tag{3.7}$$

where $0 < t \leq 1$ is an input parameter to represent the proportion of walkers that move on to take another step. This will be referred to as the *damped* embedding, as it captures the random walker state damped over a range of step lengths. A range of values for t from Equation (3.6) will be systematically tested in the Section 3.2.

Random walks have many nice properties, including the an extra benefit of interpreting heavier edge weights as bringing vertices closer rather than farther away, as would be done in a standard shortest paths algorithm. This is more common in real world datasets, for example in a social network where an edge weight could indicate the frequency of two friends messaging each other, or in a food web where to indicate the strength of an interaction between two species. The weights on the graph in Figure 3.3 describe how many times characters appeared in the same scene together, and this extra information was used in the clustering process. The fact that the Markov chain matrix is a probability distribution also allows it to nat-

urally be applied to measures such as the Kullback-Leibler (KL) divergence

$$d_{ij} = \sum_k \mathbf{P}_{ik} \log \left(\frac{\mathbf{P}_{ik}}{\mathbf{P}_{jk}} \right) \tag{3.8}$$

which can be interpreted as the relative entropy of i relative to j. Another popular measure is the Wasserstein distance

$$d_{ij} = \int_{-\infty}^{\infty} |\mathbf{F}(i) - \mathbf{F}(j)| \tag{3.9}$$

where $\mathbf{F}_i$ denotes the cumulative probability distributions of i and j (Ramdas, Trillos, and Cuturi, 2017). This can be interpreted by imagining the area under the curve of one distribution as mass to be moved, and measuring the amount of work needed to transform that mass into the second distribution. Both are natural and popular dissimilarities in the context of probability that are commonly used in the field of machine learning (Goodfellow, Pouget-Abadie, et al., 2014; Arjovsky, Chintala, and Bottou, 2017). The Wasserstein distance is already a distance metric (Villani, 2003) and so can safely be used as an input. KL is not even a dissimilarity in Equation 3.8 as it is not symmetric, and so it is commonly symmetrised by adding the two directions together as $d'_{ij} = d_{ij} + d_{ji}$ (Kullback and Leibler, 1951).

One final dissimilarity based on random walks that has been applied before to clustering applications is the Euclidean commute time distance (Yen et al., 2005). This is defined as the average time it takes for a random walker to travel from a one vertex to the other and then back again. It is also a distance metric (Qiu and Hancock, 2007), and can be found in closed form using the Moore-Penrose pseudoinverse of the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$ as

$$d_{ij} = |E|(\mathbf{L}^+_{ii} + \mathbf{L}^+_{jj} - 2\mathbf{L}^+_{ij}) \tag{3.10}$$

where $\mathbf{L}^+$ is the pseudoinverse of $\mathbf{L}$ (Albano and Messinger, 2012). The distances

derived from Equations (3.4)–(3.10) will all be used in the following section as part of an experimental study to determine which is most effective.

3.2 Experimental study

Since clustering is an undefined problem, the quality of clusters will be measured by using a benchmark set of graphs where a ground truth clustering configuration is known.

It is important to note that the contributions in this chapter are solely in comparing different hierarchical clustering methods by quantifying the choice between them. This means that the edge bundling problem has been reduced to the task of hierarchical clustering in a network context. This means that a comparison to other edge bundling methods is out of scope, but also means that the results presented have potential significance in the general field of network clustering as well as for bundling.

Table 3.1 contains the four graphs used in this study. The graph `karate` is a classic dataset that has been extensively studied in the network research literature (Fortunato and Hric, 2016), originally compiled by Zachary (1977). It is a social network containing relationships between members of a karate club, in which a rift between two club leaders caused the club to split into two new clubs, where the resulting members of the two clubs form the ground truth clustering of the data. The `footballTSE` graph is a network of American football games between Colleges in the US, where the ground truth is the conference in which they belong. It was originally studied by Girvan and Newman (2002) in the same paper as the GN algorithm was presented. The ground truth data used is a version corrected by Evans (2010), as the original data clusters were assigned from the 2001 season

Table 3.1: Benchmark graphs used for the experiment in Section 3.2. $|V|$ denotes the number of vertices, $|E|$ the number of edges, and $|M|$ the number of ground truth clusters.

Name	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert M\rvert$
`karate`	34	78	2
`footballTSE`	115	613	19
`caltech`	762	16651	9
`lfr`	250	472	10

instead of the correct 2000 season. The `caltech` graph is a social network of students at the California Institute of Technology, extracted from the website Facebook and originally studied by Traud, Mucha, and Porter (2012). It has been shown to demonstrate clique structure (Nocaj, Ortmann, and Brandes, 2015), where the ground truth used here are the dorms in which students were living. This is the most messy data and so the dorm ground truth is the least well matched by the clustering algorithms used.

The `lfr` graph is an artificial graph from the Lancichinetti Fortunato Radicchi (LFR) benchmark (Lancichinetti, Fortunato, and Radicchi, 2008) which is a model to generate graphs with a clustered structure, given a number of parameters to account for power law distributions in cluster size and vertex degree. The parameters used here were $\tau_1 = 3$, $\tau_2 = 1.5$, and $\mu = 0.1$. They represent the degree distribution, community size, and fraction of edges to place between communities, respectively. The LFR benchmark was developed in order to improve the popular random model of Girvan and Newman (2002). All graphs are undirected and unweighted.

These four graphs were run through each of the agglomerative methods presented in Section 3.1.3. For methods that first require a high dimensional embedding, a range of parameters were used for the values of t for Equations (3.6) and (3.7). To assess the quality of the results, the adjusted random index (ARI) was used to com-

pare a ground truth clustering against the clusters predicted by the algorithm. The benefit of using ARI is that it is robust to cluster labels being swapped, avoiding the situation where the clusters are correct but the order of labels is not. ARI is bounded between -1 and 1, where a value of 1 denotes perfect clustering, 0 denotes a randomly shuffled clustering, and -1 denotes an even worse match than the expected result of a shuffle. A problem with hierarchical clustering is that, to extract flat clusters to match the ground truth, the resulting dendrogram must be cut. The results here give the benefit of the doubt by measuring ARI at all cutting points and taking the maximum ARI of all cuts, to essentially assume that the algorithm always chooses its cut perfectly.

The package `scipy.cluster.hierarchy` (Virtanen et al., 2020) was used to perform all hierarchical dissimilarity-based clustering. Results are presented in Figures 3.4 and 3.5, where all dissimilarity measures are clustered with each of the five linkage methods presented in Section 3.1.3.

3.2.1 Results

The aim of this study is to answer two questions: which dissimilarity measure is best, and which linkage method is best for clustering this dissimilarity? Unfortunately there is no clear-cut answer to either question, but the results can be interpreted to give some recommendations. Looking at the bottom right panels for each graph, it is immediately clear that single linkage is the least effective and should be avoided. The best scores for the remaining linkage methods generally hover around the same value, and so it is clear that the remaining four linkage methods all *can* result in high quality results. Looking at the mean ARI scores, however, it becomes apparent that Ward's method produces the best results on average. This is consistent with the choice made by Pons and Latapy (2006) for

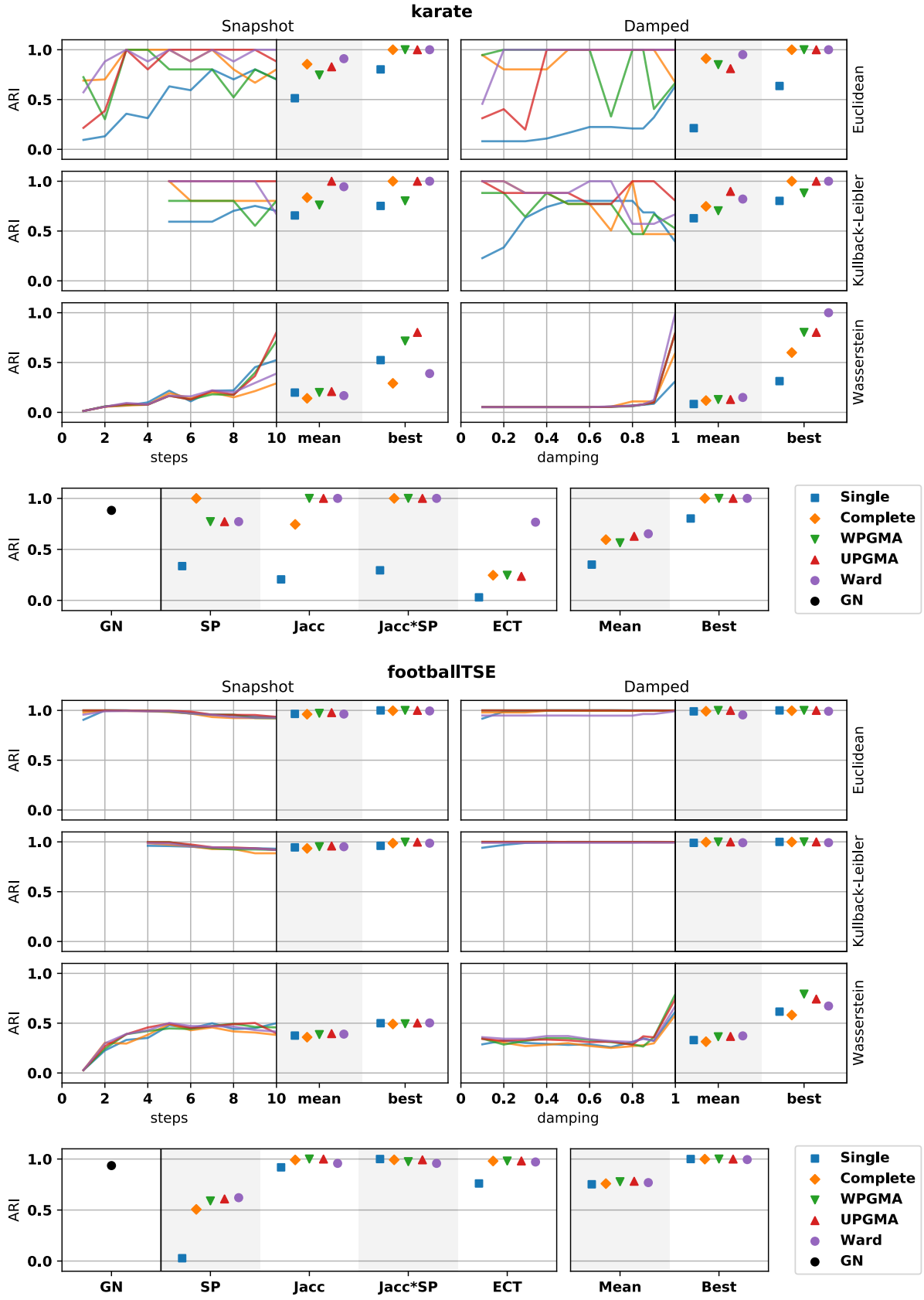


Figure 3.4: Clustering performance on the graphs `karate` and `footballTSE`. Each linkage method is denoted by a different color and symbol. ARI is on y-axis, clustering algorithms on the x-axis. Average and best ARI over all clusterings is in bottom right. Missing values for KL are due to probabilities of 0 in the Markov chain.

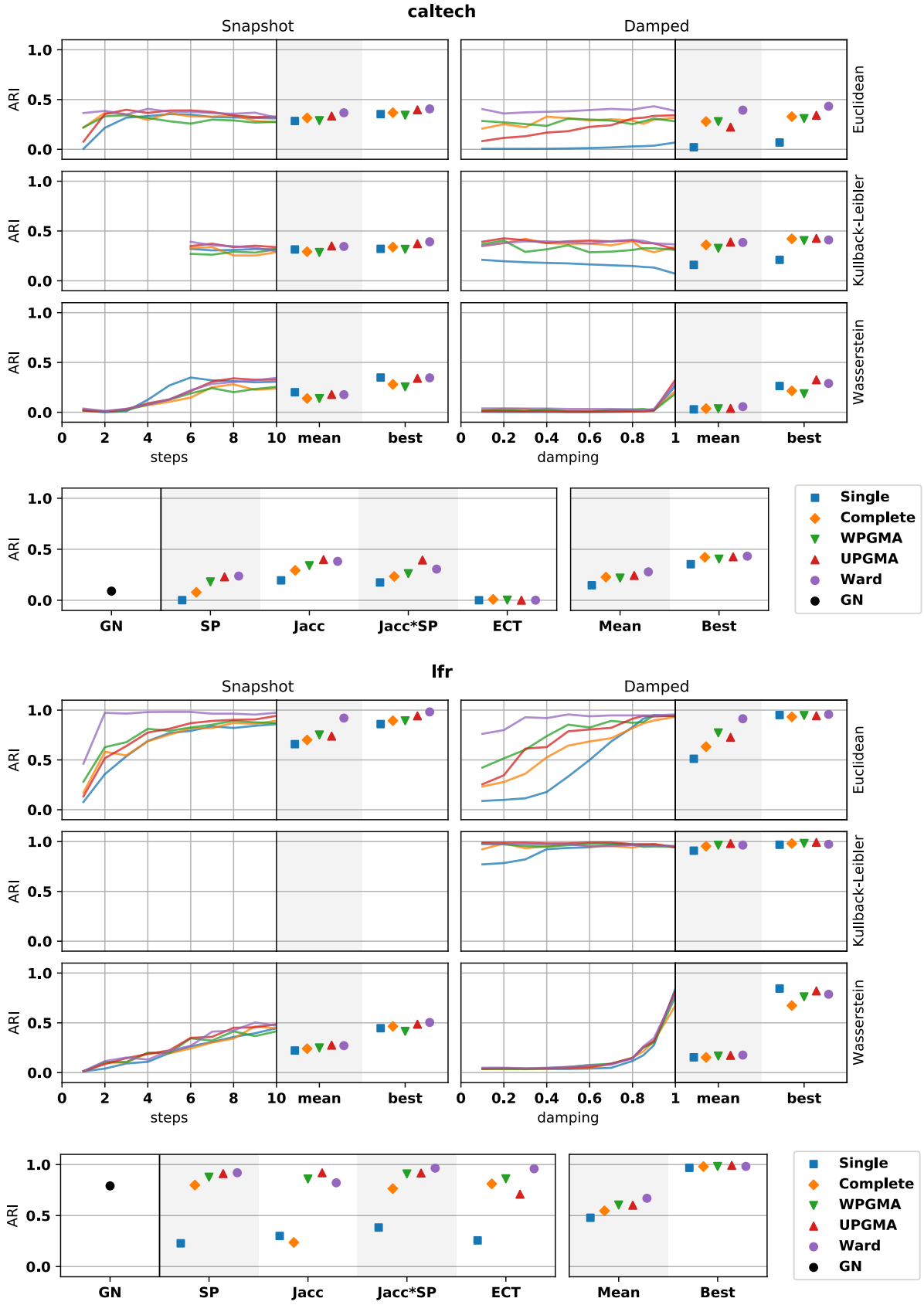


Figure 3.5: Clustering performance on the graphs `caltech` and `lfr`, following the same format as Figure 3.4.

their clustering method, now with quantitative results to verify the decision.

The best dissimilarity for Ward over all graphs is Euclidean on snapshot embedding with $t = 5$ in Equation (3.6), and it seems the original authors in Pons and Latapy (2006) agree with this choice too, as they performed their analysis using either $t = 2$ or $t = 5$. Using Equation 3.7 to take a weighted average of step lengths also does not improve the clustering performance, further confirming their design choices. Despite being a distance metric, the performance of Wasserstein is consistently poorer than both Euclidean and KL. It is however worth noting that it consistently improves as the value of t rises for both snapshot and damped embeddings. Both sets of parameters converge to the same embedding (the stationary distribution of the Markov chain) for $t \to \infty$ and $t \to 1$, respectively, so it seems that Wasserstein is effective on this stationary distribution, at least on this benchmark.

The alternative dissimilarities in the bottom left panels do not reach the same best values as the embeddings do, but they possess the important benefit of not needing to tune an extra parameter t, which is of great importance to practitioners. This is especially true given the variance of some of the lines in the top plots in Figures 3.4 and 3.5. The Jaccard index combined with UPGMA consistently scored high in each of the test graphs, and so is also worth a recommendation as an interpretable and robust choice. The remaining results in this chapter will use Ward's method on Euclidean distance over a snapshot embedding with $t = 5$, due to the reasons outlined above.

3.2.2 Pruning

The final step to producing a hierarchical edge bundling visualisation is to convert the resulting dendrogram into an edge bundled diagram. This is done by first lay-

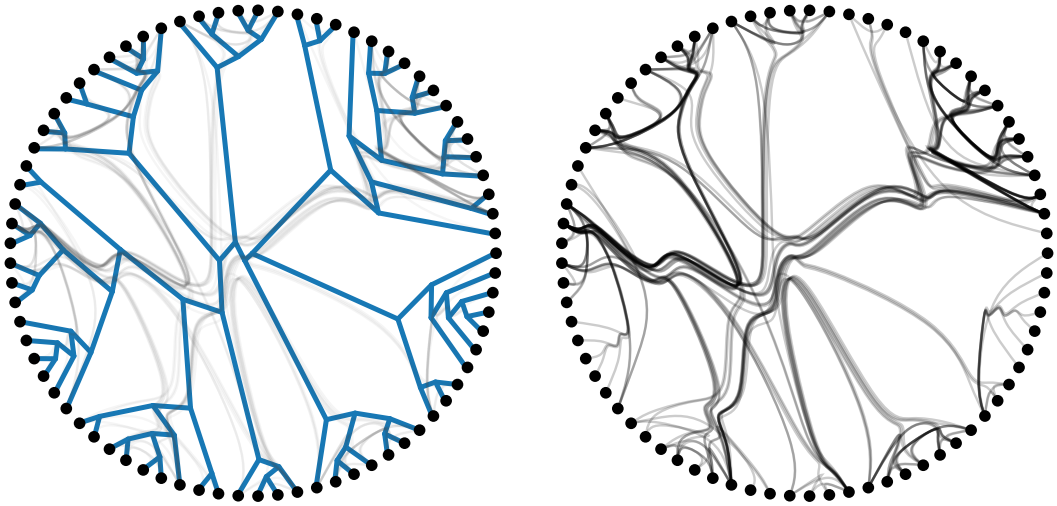


Figure 3.6: An example of the resulting hierarchical edge bundles when pruning is not applied to the hierarchy. On the left is the full hierarchical dendrogram without pruning, and on the right is the resulting bundled diagram, which is clearly worse than the one in Figure 3.3.

ing out the hierarchy as a tree, and then redrawing the original edges back on top as splines routed up and down the hierarchy. However a problem with using a dendrogram is that, as a binary tree, it contains as many branch nodes as leaves, which means that the resulting splines end up being routed through too many control points to separate individual edges. See Figure 3.6 for an example of this on the `lesmis` graph. The process of tidying up this tree such that it is suitable for use in hierarchical bundling will be referred to as *pruning*.

Jia, Garland, and Hart (2011) also encountered the same problem when producing their hierarchical bundles using the resulting dendrogram from the GN algorithm. They alleviate this issue by applying a branch-merging procedure that leverages the fact that each branch corresponds to a disconnected subgraph, due to the progressive removal of edges at each step in the algorithm. This cannot be applied to a general dendrogram, and so another property of dissimilarity clustering is used instead: the fact that each parent cluster can store the dissimilarity between its two child clusters before the merge. This is usually thought of as the height of each branch in the dendogram, as shown in the top-right of Figure 3.3, and is used

here to recursively merge branches in the hierarchy if their height difference falls below a certain threshold. This follows another similar idea of Pons and Latapy (2006) who use the same branch height information to determine an optimal cutting point for forming flat clusters.

The next step is to lay out the hierarchy itself as a tree. Following Holten (2006), a radial tree layout is chosen, and since each leaf of the tree represents a vertex of the original graph, they are first placed around the circumference of the circle. The order in which leaves are placed around the circle is very important, as it directly affects the proximity of edges to be bundled together. This is a similar problem to seriation in matrix plots (Liiv, 2010). Obviously any cousin vertices in the tree should be placed next to each other, but this still leaves an exponential number of orderings because the order within groups of cousins is still undecided. Fortunately, a neat solution exists to this problem, as there exists an algorithm that can find an optimal ordering. The definition of optimal in this case is to minimise the total dissimilarity between leaves ordered next to each other. Remarkably, this can be computed in polynomial time using the algorithm of Bar-Joseph, Gifford, and Jaakkola (2001), which leverages the structure of the hierarchical dendrogram to formulate the task as a dynamic programming problem. All resulting layouts presented here use this optimal ordering, although its complexity of $\mathcal{O}(|V|^4)$ makes it prohibitively slow for datasets any larger than the graphs studied here.

The next step is to position the branches of the tree. Many methods were tested for determining branch placement, including the use of force-directed methods such as Tutte's algorithm as in Figure 2.3, but the method settled upon was to use polar coordinates, in the following manner. The angle of a parent is set to the mean of the angles of its children, and its distance from center is set proportional to the total number of leaves below it in the hierarchy. This means that the height

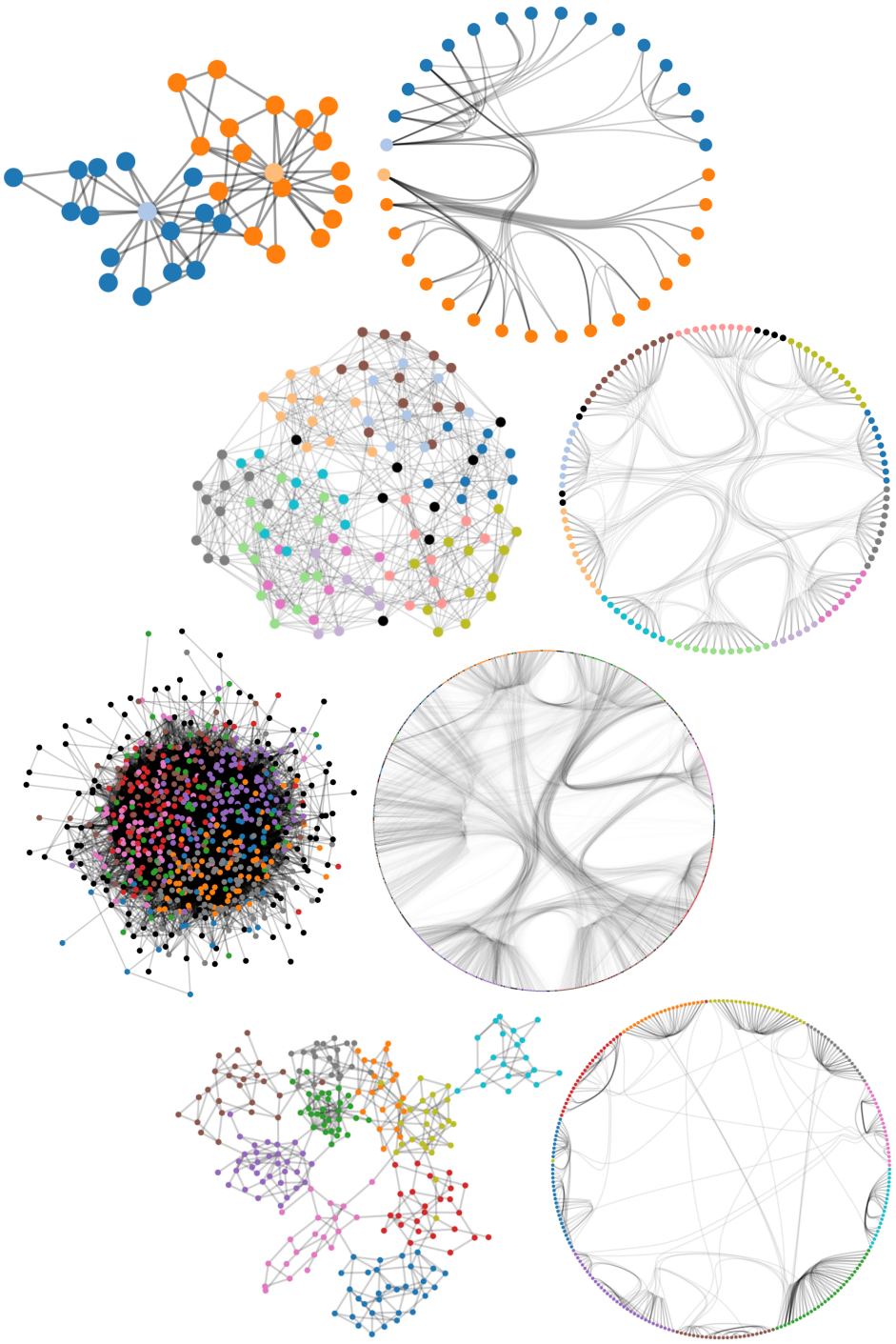


Figure 3.7: Drawings of the graphs in Table 3.1 using both a stress layout (left) and with hierarchical edge bundling (right), ordered in the same sequence as in the table. Colours of nodes correspond to the ground truth clusters included in the data. In `karate`, the two leaders of the club are coloured lighter. For `football`, 8 of its teams are singleton clusters in independent divisions, coloured black. For `caltech`, 168 students are unassigned to dorms, also coloured black. For all graphs, a branch merge distance of 10% of the dendrogram height was used, with merged branches placed at a polar distance equal to the fraction of leaves it has as children, cubed.

of a branch corresponds to the size of its corresponding cluster, which is extra information embedded in the final visualisation.

The final step is to render the bundles itself. With the tree layout determined, this step is exactly the same as that of Holten (2006). Important subtleties include relaxing bundles to prevent splines from overlapping too much, and skipping the lowest common ancestor as a control point in order to reduce clutter around common parent nodes. The resulting visualisations for the four benchmark graphs in Table 3.1 can be seen in Figure 3.7. Code used to generate all figures can be found at `www.github.com/jxz12/HierarchicalEdgeBundles`.

3.3 Discussion

The work done in this chapter presents a pipeline for producing hierarchical edge bundles (Holten, 2006) on general graphs without hierarchical metadata available, as illustrated in Figure 3.3. This was done by agglomerative hierarchical clustering, a method widely available in many data analysis software packages. A range of parameterisations for performing this clustering were compared, and some practical recommendations will be further elaborated in this final section.

In general, the results presented largely agree with the design choices made by Pons and Latapy (2006) for the use of agglomerative dissimilarity clustering on graphs. That is except for leaving out the extra $\mathbf{D}_{kk}^{-1/2}$ weighting in Equation 3.6 as it did not improve performance at all in the benchmark explored here. The use of dissimilarity measures based on random walks also outperforms those derived from other features such as shortest paths, as well as the GN algorithm used by Jia, Garland, and Hart (2011), which also has a higher runtime complexity. Calculating the betweenness-centrality for each edge is $\mathcal{O}(|V||E|)$ (Brandes, 2001a) and since

this must be recalculated $|E|$ times, the overall complexity of GN is $\mathcal{O}(|V||E|^2)$, compared to $\mathcal{O}(|V|^2 \log |V|)$ for the agglomerative methods explored here.

Ward's method also consistently outperforms other linkage methods, and a qualitative observation of note is that it also tends to give dendrograms that lend themselves more to the pruning process in Section 3.2.2. Branches in Ward dendrograms more clearly delineate separately groups of clusters, leading to easier simplification into a streamlined hierarchy for bundling. When the opposite situation occurs, i.e. if differences in branch height are homogenous, the pruning process may merge every single branch into one, since child branches are recursively merged into their parents. However this also usually corresponds to a poor quality clustering in general, because the algorithm could not find large groups of datapoints all close to each other.

In a real-world visualisation pipeline, this pruning process would ideally be human driven, with methods such as MLCut of Vogogias et al. (2016) allowing dendrograms to be interactively explored and cut to form flat clusters. Such a tool could be easily applied to perform multiple cuts in order to form a hierarchy for bundling. Pruning is also not the only way to solve the dendrogram issues of Figure 3.3. For example, different types of splines may not result in the same issues. A more thorough examination into the use of B-splines can be found in Section 4.2.5.

Interaction would also greatly improve the utility of these visualisations. Holten (2006) suggested ideas such as dynamically tweaking the amount that bundles are relaxed, and highlighting interesting bundles by hiding other links. An additional possibility in this context would be to interactively prune the hierarchical tree in order to neaten bundles, or being switching between different clustering algorithms to reveal different patterns depending on the algorithm selected. The need for interaction can also be seen in the complex final render `caltech` in Figure 3.7, second-from-bottom; a dynamic option such as *GrouseFlocks* by Archam-

bault, Munzner, and Auber (2008) would be useful here.

A quantitative assessment of the actual utility of resulting bundles is something missing from the literature, but out of scope for the work here. It is however clear that certain topological features are made more salient by these diagrams. A good example of this is can be seen in the `karate` graph in Figure 3.7, where it is clear that the two nodes on the left are central to the topology of the graph, due to the thickness of the bundles stemming from them. These two are in fact the two club leaders whose disagreement caused the club to split into the two factions used as ground truth for clustering. The algorithm was able to place them next to each other in a completely unsupervised manner, due to the quality of the extracted hierarchy combined with the optimal ordering (Bar-Joseph, Gifford, and Jaakkola, 2001) outlined in Section 3.2.2. A possible study to quantify the effectiveness of bundles could be to ask participants to identify ground truth cliques or bicliques from different bundling configurations.

Future work could also include testing on a larger benchmark, especially with graphs with more biclique than clique structure. The current benchmark also measures the quality of clustering, but not bundling directly. Such a benchmark could be used to compare the hierarchical bundling method here directly with other (non-hierarchical) methods. In a similar vein, a dissimilarity measure or objective function that quantifies bundling quality could be directly optimised to produce high-quality bundles.

Chapter 4

Power-confluent drawing

Edge bundling can be likened to data compression on graphs, as the goal is to describe a given network without having to know the source and target for every individual edge. Such compression would be impossible in a random network, but because real-world data does have structure, underlying patterns can be leveraged to group together similar components within the data.

The hierarchical edge bundling of the previous chapter can be interpreted as *lossy* compression, because the grouping of edges can obscure the exact endpoints of each edge. In the same vein as, say, JPEG image compression, there exists a tradeoff between the amount of data used against the amount of information displayed. But just as there exist *lossless* compression formats for images, such as PNG compression, there also exists such a counterpart for edge bundling, known as *confluent drawing*. An example of such a drawing can be seen on the right-hand side of Figure 4.1, where the bundles are interpreted as follows: every possible smooth path from one vertex to another indicates that they are connected by an edge in the original graph.

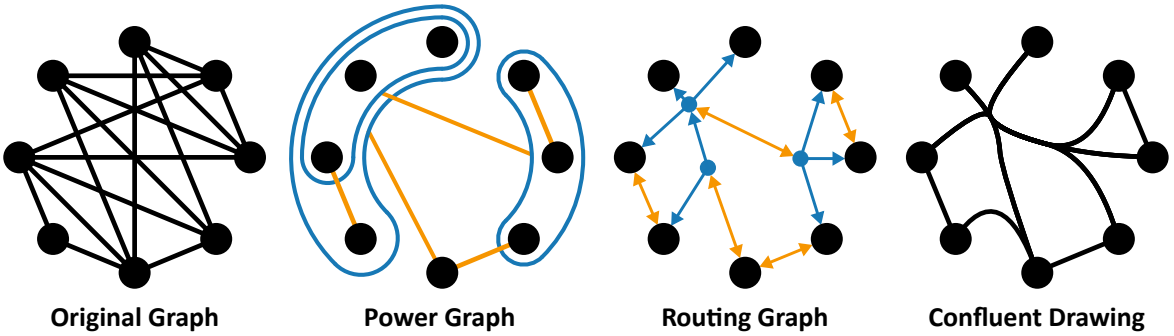


Figure 4.1: A pipeline for producing power-confluent drawings. From left to right: a conventional node-link diagram with vertices arranged around a circle; its power graph decomposition, where edges are compressed by grouping similar verticess together; a novel equivalent routing graph, where the nested structure of groups is represented by a directed tree; the resulting confluent drawing, where original edges are threaded through the routing graph to form bundled junctions.

4.1 Background

Confluent drawing has its roots in the more theoretical side of graph drawing, and was originally developed by Dickerson et al. (2005) as a novel method for drawing planar graphs. In Figure 4.1 for example, the drawing of the original graph on the left contains many crossings and in fact can not be drawn in a planar way. Allowing edges to overlap, but requiring all smooth curves to denote an edge between endpoints, means that all connectivity is preserved, thus adhering to the analogy of lossless compression.

This process is analogous to inserting junctions on a train track to allow carriages to switch directions without stopping. The precise definition of a confluent drawing will be outlined later in Section 4.1.2.

The work in this chapter will study a subset of confluent drawing known as *power-confluent* drawing. While a standard confluent drawing needs only to follow the definition in Section 4.1.2, with no restrictions on its creation process, producing a power-confluent drawing consists of three steps: finding the *power graph decomposition;* converting this decomposition into an equivalent auxiliary routing

graph (ARG), similarly to hierarchical edge bundling in Section 3.2.2; and finally drawing curves back on top to render a confluent drawing. Each step is illustrated in Figure 4.1. This was an idea originally explored by Bach et al. (2017), and the work presented here will aim to further develop and formalise the implementation of this idea. Furthermore, it will be proved in Section 4.2.7 that this specific process can only ever result in a subset of all confluent drawings, thus earning the terminology of power-confluent.

4.1.1 Power graph decomposition

The first step to producing a power-confluent drawing is to construct a power graph, and so this step will be outlined first. A power graph is an extension to the conventional node-link diagram, that allows vertices to be grouped together into *power groups*, in order to merge edges that share endpoints into individual *power edges* (see the leftmost two drawings in Figure 4.1). Power groups may also be nested in a hierarchical structure, allowing power edges to cross group boundaries.

This idea was originally developed for use in biological protein–protein interaction network analysis by Royer et al. (2008), who originally computed a decomposition by using heuristics based on the Jaccard index, defined here in Equation 3.4. This was later improved by Dwyer et al. (2014) who presented multiple methods of finding both optimal and sub-optimal decompositions, while also proving that the problem in general is likely to be $\mathcal{NP}$-hard. The method they recommended for practical use is a greedy agglomeration technique, not unlike the ones studied in Section 3.1.3, except without the benefit of the Lance-Williams property shown in Equation (3.3). Its complexity is $\mathcal{O}(|V|^3|E|\log|E|)$, as it must search through $\mathcal{O}(|V|^2)$ merges $\mathcal{O}(|V|)$ times, and each merge takes $\mathcal{O}(|E|\log|E|)$ time to assess

its quality. In Section 4.2.1 this assessment step will be improved slightly to $\mathcal{O}(|E|)$, and in practice the number of edges to be considered diminishes quickly anyway (Dwyer et al., 2014).

This is also the method chosen by Bach et al. (2017), to construct their initial power graphs, and so is the one studied here. Note that the following description is specific to undirected graphs; the changes required for directed graphs are described in Section 4.2.6.

Greedy agglomeration

The algorithm works taking advantage of the fact that the hierarchy of power groups can also be represented as a tree, with vertices as the leaves, similarly to the hierarchical clustering of Section 3.1.3. It first initializes every vertex as the sole member of a power group, henceforth referred to as a *module*, and initializes a set of neighbours for each trivial module based on the edges of the graph. It then greedily merges pairs of top-level modules at a time, picking the merge that eliminates the most edges at each step, until no more edges can be eliminated. The number of eliminated edges for any given merge is given by

$$\kappa_{\cap}(m, n) = |N(m) \cap N(n)| \tag{4.1}$$

where m and n are modules and N are their neighbour sets. This is almost the same as $\mathrm{nedges}(m, n)$ in Dwyer et al. (2014), except as a positive score rather than a negative penalty.

This is the stage where the difference between this power decomposition and hierarchical clustering becomes clear: since power decomposition must preserve connectivity information, in line with the lossless compression analogy, care must be taken when performing the merging step in order to keep edges and power

edges pointing at the correct endpoints. The simplest case is when two modules m and n share the exact same neighbours, i.e. $N(m) = N(n)$. In this case one module can simply be 'absorbed' into the other, such that every pair of edges is merged into one, and the number of modules decreases by one.

A trickier case is if only one of the modules contains edges that cannot be merged. For example, m cannot fully absorb n if $N(m) \supset N(n)$. Leaf modules of the hierarchical tree, i.e. individual vertices, also cannot be absorbed, and so the same situation occurs when m is a leaf and n is a module. In both cases m adopts n, and $N(m) \cap N(n)$ is removed from $N(n)$. If both contain edges that cannot be absorbed, i.e. $N(m) \not\subset N(n)$ and $N(m) \not\supset N(n)$, then a new module is created that adopts m and n as children and steals $N(m) \cap N(n)$ as its neighbour set. This also happens if m and n are both leaf modules. Note that this is done equivalently in the original description by always generating a new parent module, and removing either child if their neighbour sets become empty (Dwyer et al., 2014). The greedy process above will be the subject of improvements in Section 4.2.1.

4.1.2 Confluent drawing

The original definition of a confluent drawing A for a graph G is:

- There is a one-to-one mapping between the vertices in G and A, so that, for each vertex $v \in V(G)$, there is a corresponding vertex $v' \in A$, which has a unique point placement in the plane.
- There is an edge $(v_i, v_j) \in E(G)$ iff there is a locally-monotone curve [defined as having no self intersections or sharp turns] e', connecting v'_i and v'_j in A.
- A is planar. That is, while locally-monotone curves in A can share overlapping portions, no two can cross.

This was proposed by Dickerson et al. (2005), where the second condition is the key part of the definition that lets links overlap in the final drawing. It is not immediately obvious how to convert a power graph into such a drawing. Fortunately, Bach et al. (2017) proposed the idea of first converting a power graph decomposition into an ARG. Then, for each adjacency, the graph-theoretic shortest path through the ARG is used as the sequence of control points for a B-spline (Sederberg, 2005).

This process is illustrated in Figure 4.1, and the ARG used is almost exactly the one in hierarchical clustering (Section 3.1.1), where each layer of the hierarchy is mapped to a branch vertex in a hierarchical tree. The only differences are the removal of the root of the tree, and the addition of extra edges between tree vertices to represent power edges, as shown in Figure 4.1, second from right. Note that in the original algorithm edges in the ARG are undirected (Bach et al., 2017), unlike those shown in Figure 4.1. The addition of directed edges to the ARG will be used to solve various theoretical issues in Section 4.2.4.

4.2 Improvements

This section will describe multiple improvements to the methods described in the previous section, which encapsulate the novel contributions from this chapter. The speed and quality of the power decomposition algorithm will be improved in Section 4.2.1, and two theoretical issues will be resolved for the conversion to a confluent drawing in Sections 4.2.2 and 4.2.3.

Specifically for the conversion, it will be shown that the combination of B-splines and shortest paths introduces problems not fully explored by Bach et al. (2017), causing the resulting drawings to violate the second and third conditions in the

above definition. These problems will be solved, and the subclass of confluent drawings the algorithm can produce will be identified as a new classification known as *power-confluent*. The corrected solution guarantees that the second condition of the definition in Section 4.1.2 is always satisfied, but still cannot guarantee the third, as discussed in Section 4.2.7.

It is important to note that these theoretical issues are rare in practical use cases of Bach et al. (2017), and that the solutions proposed here will not often change the resulting drawings. When the issue does occur however, it changes the implied connectivity of the graph. Ensuring correctness is also necessary for work on the theoretical side of the method to further progress.

4.2.1 Greedy heuristics

The algorithm outlined in Section 4.1.1 greedily selects the next merge based on the reduction in edges as in Equation (4.1). However this is not the only metric that the greedy algorithm can use to judge the quality of any given merge. Only using one heuristic often results in many merges with the same score, and so the choice of merge may be arbitrarily chosen from many candidates, some of which may lead directly to local optima. In such situations it can be useful to include additional heuristics to further discriminate between choices. This is not explicitly mentioned in Dwyer et al. (2014), but is implemented in their provided source code, where they include two other metrics: minimising the total number of groups/modules after the merge, and minimising the number of times power group boundaries are crossed by power edges.

Here a new heuristic will be introduced that can roughly capture the effects of both, and is simpler to calculate: a penalty for the number of edges that could not

be merged. This is defined by

$$\kappa_{\triangle}(m,n) = |N(m) \triangle N(n)| \tag{4.2}$$

where $\triangle$ denotes the symmetric difference, i.e. the number of unshared neighbours, between the two sets. This effectively measures the number of edges that cannot ever be eliminated in future steps, because only top level modules are considered for merging. Equation (4.2) captures the number of modules because a new parent module is only added if both children have unshared neighbours, and so the smaller Equation (4.2) is, the fewer modules there will likely be. It also captures the effect of edges crossing group boundaries, because any unshared edges must cross the boundary of its new parent module after the merge. Rewarding the first heuristic and punishing the second leaves a total score of

$$\kappa(m,n) = w_{\cap}\kappa_{\cap}(m,n) - w_{\triangle}\kappa_{\triangle}(m,n) \tag{4.3}$$

where $w_{\cap}$ and $w_{\triangle}$ determine the relative weight of either heuristic. For example, to construct a modular decomposition (Habib, de Montgolfier, and Paul, 2004), $w_{\triangle}$ may be set to infinity to forbid any module boundary crossings. Setting $w_{\cap} = 10$ and $w_{\triangle} = 1$ works well in practice; these are the parameters used for the results in Table 4.1.

Adding the new heuristic produces the best or joint best results in all graphs, and also improves the consistency of the output, with a better worst result in all cases. Note that the variation between runs in the implementation used is due to a pseudorandom order of iteration through candidate merges[1] (Figure. 4.2, line 4) and so when multiple merges have the same score, the choice becomes nondeterministic. The networks are, from top to bottom, Italian families linked by marriage

[1]This is due to the use of `std::unordered_set` in C++, which may vary between implementations of the standard library (Josuttis, 2012).

Table 4.1: Experimental results for the improved greedy power graph decomposition described in Section 4.1.1 $|E|$ is the number of edges in the original graph, compressed into $|P|$ power edges and $|M|$ power groups; shaded cells indicate the better score between the two methods.

Name ($\|E\|$)	Best $\|P\|$ ($\|M\|$)		Worst $\|P\|$ ($\|M\|$)	
	only $\kappa_\cap$	$\kappa_\cap$ and $\kappa_\triangle$	only $\kappa_\cap$	$\kappa_\cap$ and $\kappa_\triangle$
`florentine` (20)	11 (4)	11 (4)	11 (6)	11 (5)
`karate` (78)	28 (13)	28 (13)	30 (15)	29 (13)
`southern` (89)	30 (17)	27 (21)	37 (16)	30 (18)
`dolphins` (159)	82 (29)	81 (30)	87 (28)	83 (30)
`lesmis` (254)	74 (39)	72 (41)	79 (39)	72 (42)
`football` (613)	282 (83)	278 (84)	289 (83)	286 (84)
`netsci` (914)	355 (187)	338 (184)	371 (189)	341 (186)

(Breiger and Pattison, 1986), members of a karate club (Zachary, 1977), women meeting at social events (Davis, Gardner, and Gardner, 2009), interactions between bottlenose dolphins (Lusseau et al., 2003), co-occurrence of characters in the musical Les Misérables (Knuth, 1993), American football games between US colleges (Girvan and Newman, 2002), and coauthorships of scientists working on network theory (Newman, 2006a).

It is important to note that this is still a simple greedy heuristic, which does not make any guarantees about the optimality of the final result. The original method by Dwyer et al. (2014) gives the option to somewhat alleviate this, by optionally maintaining a priority queue of the best configurations seen so far, along with some dynamic programming to prevent re-evaluating configurations already seen. A further exploration of this extended algorithm is out of scope for this paper, and Dwyer et al. (2014) additionally note that the qualitative improvement that results from including this priority queue is minimal.

The final change involves bringing down the complexity of neighbour set intersection to $O(E)$ using hash sets, which reduces the complexity of the algorithm down slightly to $O(|V|^3|E|)$. Pseudocode for the above described algorithm can be seen

Algorithm 3: GREEDY POWER GRAPH DECOMPOSITION

inputs : graph $G = (V, E)$
output: modules M = set of pairs (C, N)
1 $M \leftarrow \{(\varnothing,\ N(v)) \mid v \in V\}$
2 **do**
3 $\quad \kappa_{\text{best}} \leftarrow 0$
4 $\quad$ **foreach** pair of modules $\{m, n\} \in M_{\text{top}} \times M_{\text{top}}$
5 $\quad\quad \kappa_{\text{best}} \leftarrow \max(\kappa(m, n),\ \kappa_{\text{best}})$
6 $\quad$ MERGE$(\{m, n\}_{\text{best}})$
7 **while** $\kappa_{\text{best}} > 0$

Figure 4.2: Pseudocode for the greedy heuristic power graph construction in Section 4.1.1. Each module consists of a set of children C, and a set of neighbours N. The score function κ is from Equation (4.3), and the merge operation on line 6 is described in Section 4.1.1, where any new module is parented to its merged children by adding them to its set of children C. In practice a redundant super-module is maintained, whose children are the top level modules M_{top} on line 4.

in Figure 4.2.

4.2.2 Node splitting

The improvements described will now move onto the conversion of the power graph into a power-confluent drawing. Issues regarding the use of B-splines are first identified (of degree $p = 3$ in the source code of Bach et al. (2017), although it is not specified in the paper) for interpolating control points. These were likely chosen because they satisfy the convex hull property, which prevents crossings at shared control points. They also offer local control (i.e. moving a control point only affects the surrounding $p + 1$ segments), which guarantees that splines that share enough intermediate control points will overlap.

Local control is what makes it possible for drawings to be confluent; with the right routing graph, it is possible for edges to share enough control points to produce the overlapping portions that are required for a confluent drawing. Specifically, for two curves to be guaranteed to overlap they must share p or more control

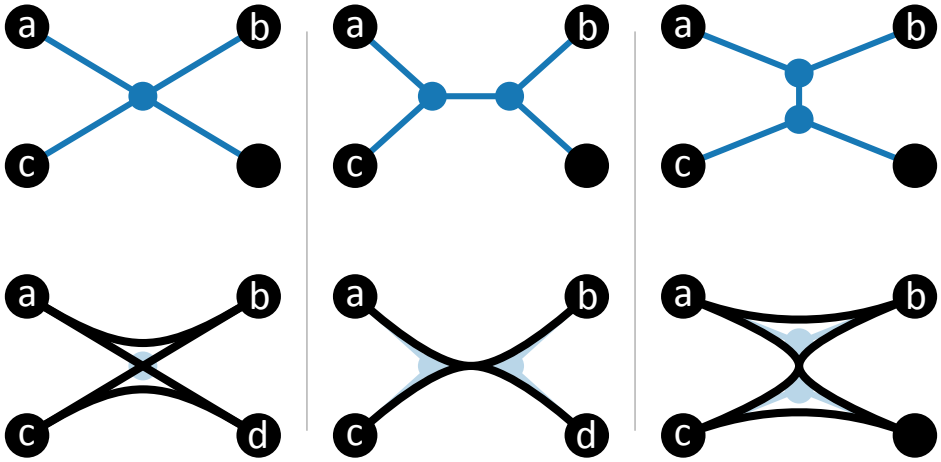


Figure 4.3: Examples to show how the direction of node splitting can affect the graph $K_{2,2}$. The first column shows a standard situation where a split is required to make edges overlap, the second shows the correct split, and the third shows that if the node is split the other way, then the edges $\{a, c\}$ and $\{b, d\}$ are falsely introduced.

points (see Section 4.2.5 for a proof and a more detailed explanation).

There are two problems with using B-splines in this context. The first is that splines that share fewer than p control points will *not* overlap, but sharing even a single routing node should indicate a bundled junction. The authors recognized this, calling it the 'crossing artifact' (Bach et al., 2017, Fig. 4), and fixed it by splitting routing nodes into two: one for incoming and one for outgoing edges. This is illustrated in Figure 4.3. The intuition behind this splitting is correct, as it introduces another shared control point that tightens the bundle, joining the crossed edges into a junction. However, their exact description contains an ambiguity in the context of undirected graphs, as it is not specified how to identify edges as incoming or outgoing.

This is problematic as an incorrect split may introduce errors into the resulting drawing, as shown in Figure 4.3, right. This ambiguity will be resolved, along with the following related problem, in Section 4.2.4.

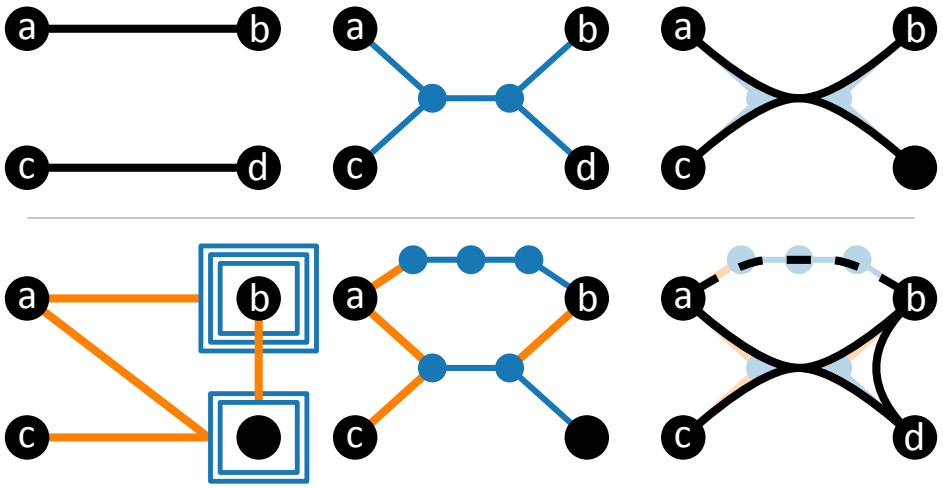


Figure 4.4: Examples of how absent edges can appear to exist due to the short-circuit problem. Top: a toy example of a hypothetical routing graph that causes edges $\{a, d\}$ and $\{c, b\}$ to both erroneously appear to exist. Bottom: an example of a power graph that causes a similar ambiguity, where c appears connected to b because the edge $\{b, d\}$ short-circuits the nested structure of power groups (represented by blue boxes), causing $\{a, b\}$ to be routed incorrectly downwards. The correct direction that would not cause an issue is shown as a dashed line along the top.

4.2.3 Short-circuits

The second problem is also caused by local control but has an opposite result: that splines will *always* overlap if they share p or more control points. For some ARGs, such as the examples in Figure 4.4, splines may overlap so as to create the visual impression that extra edges, not present in the original graph, exist. This violates the second condition in the confluent definition.

Here it will be explained why routing edges through their graph-theoretic shortest paths in the ARG can introduce false adjacencies. A power decomposition is converted into an ARG by (a) connecting the members of each power group to a new vertex corresponding to the group, and (b) connecting pairs of vertices whose corresponding power groups are connected by power edges. Since the original edges do not exist in this auxiliary graph, they are instead drawn back on top by finding the graph-theoretic shortest path between the vertices on either

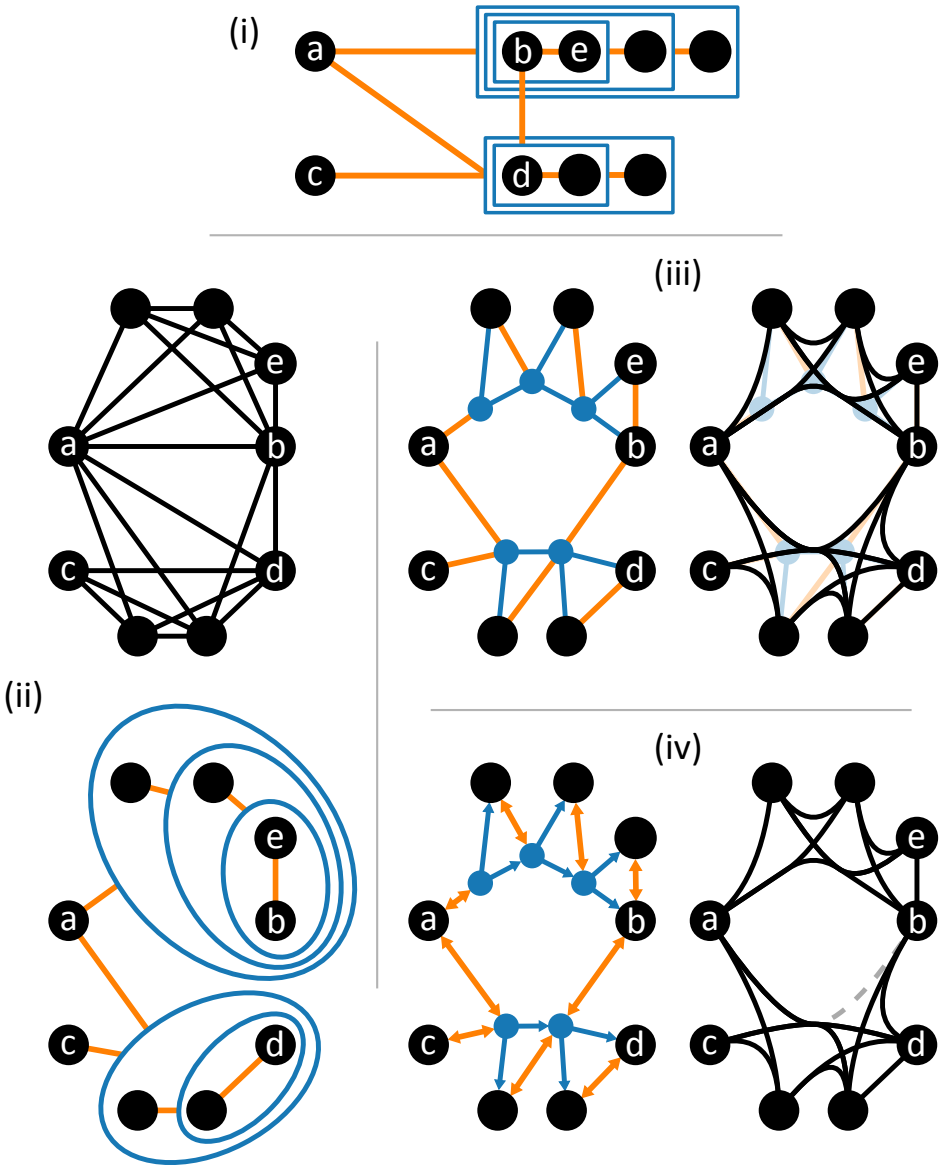


Figure 4.5: A fleshed-out version of Figure 4.4, bottom, without any redundant power groups. (i): an example of the power graph in the same layout as Figure 4.4, where the nested structure of groups comes from a clique structure. (ii) top: a conventional node-link layout of the graph, bottom: the same power graph as in (i). (iii) left: the resulting routing graph from the method of Bach et al. (2017), right: the resulting drawing when splines take their shortest paths through this routing graph. The edge $\{a, b\}$ is routed downwards in the wrong direction, causing the edge $\{c, b\}$ to falsely appear to exist. The edge $\{a, e\}$ also has two equal-length shortest paths, either through the line shown or downwards through b. (iv) left: the result of using the new method (Section 4.2.4) of retaining the hierarchical structure of power groups through directed edges, right: the resulting drawing where $\{a, b\}$ is routed through the three correct upward routing nodes. The previously incorrect routing is marked by a dashed line.

end of the edge, using the nodes on this path as the control points for a spline. However, since (a) and (b) both result in edges in the routing graph, with nothing to differentiate between them, this can cause a *short-circuit* effect, that potentially introduces false adjacencies into the resulting drawing. This is shown in Figure 4.4, where two intermediate routing nodes are used because that is the minimum number of shared control points required for quadratic splines to overlap (see Section 4.2.5). Note that every power group in this example is intentionally redundant for clarity; a full version of this example can be seen in Figure 4.5.

This effect can be explained as follows. The structure of groups within a power graph can be represented as a tree, where groups are represented by branches and vertices by leaves. Trees are geodetic (i.e. there exists a unique shortest path between any pair of vertices), but the connections introduced by power edges can act like bridges between branches, to invalidate this property and produce ambiguity either in the choice of path (if the shortest paths are equal) or in which edges exist at all, by routing splines in the wrong direction entirely. The bottom row in Figure 4.4 shows a simple example of how this can happen. While it may seem as if this counter-example is contrived and should not ever appear due to the redundant nested structure of power groups, a similar pattern arises from the optimal decomposition of a clique, shown in Figure 4.5. The key detail here is that only one power edge should ever be traversed for any given adjacency, which is guaranteed by the solution described in the following Section 4.2.4.

4.2.4 Hierarchical routing

The solution to these problems is to instead construct the ARG as follows:

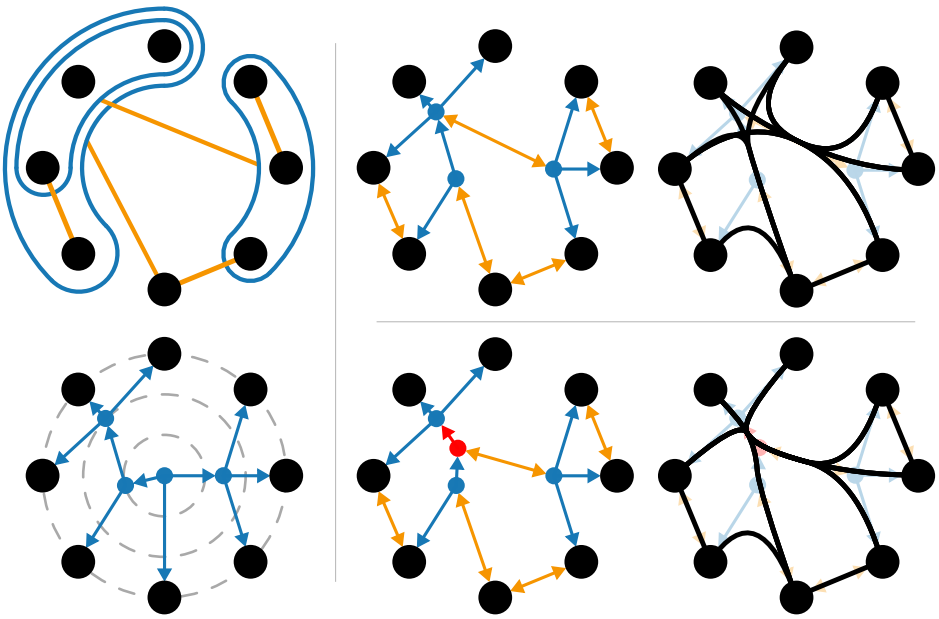


Figure 4.6: The solution to both ambiguities described in Section 4.2.4. The leftmost graphs are a power graph (top), and a tree that represents the hierarchy of power groups (bottom), but without power edges. The middle column contains the resulting routing graphs without node splitting (top) and with (bottom, split indicated in red). The final column displays the resulting bundled layouts.

- The hierarchical structure of power groups is retained as a directed tree, with all routing edges directed towards its leaves.
- Power edges are reattached between their corresponding branch vertices, as a special type of routing edge that is incoming at both branch vertices.
- The root of the tree is discarded.

This leaves the same ARG as before, except now with all routing edges explicitly directed (see Figure 4.6, middle column). Note that this also means all adjacency information is now preserved in the ARG, such that the original graph can be recovered. The purpose of having power edges outgoing at both ends will soon be made clear.

To draw the adjacency edges back on top, any shortest path calculations can now be forgoed. Instead, for each power edge, a depth-first search is performed for

Algorithm 4: POWER-CONFLUENT DRAWING

inputs : modules M = set of pairs (C, N)
output: drawing A

1 $V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $P \leftarrow \emptyset$
2 **foreach** module $m = (C_m, N_m) \in M$
3 $\quad V \leftarrow V \cup m$
4 $\quad E \leftarrow E \cup \{(m, c) \mid c \in C_m\}$
5 $\quad P \leftarrow P \cup \{(m, n) \mid n \in N_m\}$
6 **foreach** vertex $v \in V$
7 $\quad$ **if** $|N^+(v)| \geq 2$ and $|N^{-*}(v)| \geq 2$
8 $\quad\quad$ SPLIT(v)
9 $A \leftarrow \emptyset$
10 $\mathbf{X} \leftarrow$ LAYOUT$((V, E \cup P))$
11 **foreach** power edge $p = \{i, j\} \in P$
12 $\quad Q_i \leftarrow$ paths $\in (V, E)$ from leaves to i
13 $\quad Q_j \leftarrow$ paths $\in (V, E)$ from j to leaves
14 $\quad A \leftarrow A \cup \{$ SPLINE$(\mathbf{X}_q) \mid q \in Q_i \times Q_j\}$

Figure 4.7: Pseudocode for the conversion from a set of power graph modules (see the output of Algorithm 3 in Figure 4.2) to a rendered drawing, described in Section 4.2.4. Note that the edges on line 4 are ordered pairs, while those on line 5 are unordered. This allows, on line 7, for N^+ to indicate a set of outgoing neighbours, and N^{-*} a set of incoming neighbours plus any connected by power edges in P. The split operation on line 8 then moves N^+ and N^{-*} to separate routing nodes, as in Figure 4.6. In practice a shorter edge length is placed between split routing nodes, and so line 10 requires a layout method that can embed edge lengths, such as the stress layout in Section 2.1.2. The specifics of the spline function on line 14 are outlined in Section 4.2.5

all child leaf nodes starting from both ends of the power edge. Each path to each leaf from one end is then concatenated to the reversed path to each leaf from the other end, and every concatenated path is used as the sequence of control points for a spline. These paths are now guaranteed to be unique because the ARG is now effectively a tree, due to power edges being incoming at both ends to prevent their traversal. Since only the one correct power edge is traversed for each spline, the short-circuit problem described in Section 4.1.2 is alleviated.

Imposing this directionality on the routing graph also fixes the node splitting am-

biguity in Section 4.1.2, guaranteeing that the split occurs in the correct direction, according to the rule of separating incoming and outgoing edges to different nodes. Note that incoming and outgoing in this context no longer refers to edges in the original graph; how to deal with these will be described in Section 4.2.6. This works because every routing node is the boundary between two sides of a biclique, equivalent to a single bundled junction; the explicit direction of routing edges now encodes the orientation of the bundle itself. This entire process is illustrated in Figure 4.6, and pseudocode for the resulting algorithm can be seen in Figure 4.7, Algorithm 4.

4.2.5 B-splines

The choice of spline has not yet been discussed, for either confluent drawing or the hierarchical bundling of Section 3.2. The usually chosen spline in the literature is the *B-spline*, which is a generalisation of Bézier curves that offers the two nice properties of convex hull and local control, previously mentioned in Section 4.2.2. A more thorough explanation into why these properties are beneficial to bundling is, however, something generally missing from the literature, and so this section will attempt to elucidate some theory behind why they are commonly used for bundling purposes. The importance of choosing the correct spline is illustrated in Jia, Garland, and Hart (2011), in which the diagrams contain poorly implemented B-splines that do not satisfy these two properties. A good introduction to B-splines and their decomposition into segments can be found by Sederberg (2005).

It will first be proven that splines of degree p must share at least p control points to overlap. A B-spline is a parametric curve that interpolates a number of control points by summing up contributions from a basis function for each (Figure 4.8).

As a consequence, the spline is constructed as a series of polynomial curves, known as *segments*, which each satisfy the *local control* property, meaning that each segment is affected by the basis functions of only the surrounding $p + 1$ control points. These basis functions are continuous, so at the exact point at which two segments join (known as a knot) the curve cannot simultaneously be affected by the two furthest control points affecting the segments on either side. The remaining control points at the knot are therefore the last p control points of the left segment, which overlap with the first p control points of the right segment. This proof is illustrated in Figure 4.8. As a result, splines used must be quadratic, i.e. $p = 2$, for node splitting (Figure 4.3) to work as intended.

The values of knots determine the parametric intervals over which the segments span, and the above proof assumes a 'uniform' knot vector, where uniform is defined as having knots spaced along even parametric intervals. This ensures that splines with overlapping control points also have overlapping knots. To additionally ensure that splines are connected to the first and last control point, the knot vector must also be 'open', i.e. it contains p repeated knots at each end. However, there are two ways of doing this: the first retains the same total number of knots, for example a quadratic spline ($p = 2$) with 4 control points and the standard knot vector $\langle 0, 1, 2, 3, 4\rangle$ becomes $\langle 0, 0, 1, 2, 2\rangle$, which eventually reduces to a Bézier curve (Sederberg, 2005). The second appends $p - 1$ knots at either end, where $\langle 0, 1, 2, 3, 4\rangle$ becomes $\langle 0, 0, 1, 2, 3, 4, 4\rangle$, effectively duplicating the first and last control points. This second method is illustrated by the translucent curves to represent repeated knots in Figure 4.8.

In practice this second method makes the rendered curve hug its path through the routing graph closer than the first. In the case of cubic splines ($p = 3$), this means that edges come sufficiently close to look bundled, even though the above proof demonstrates that the degree must be $p = 2$ for curves to fully overlap (unless

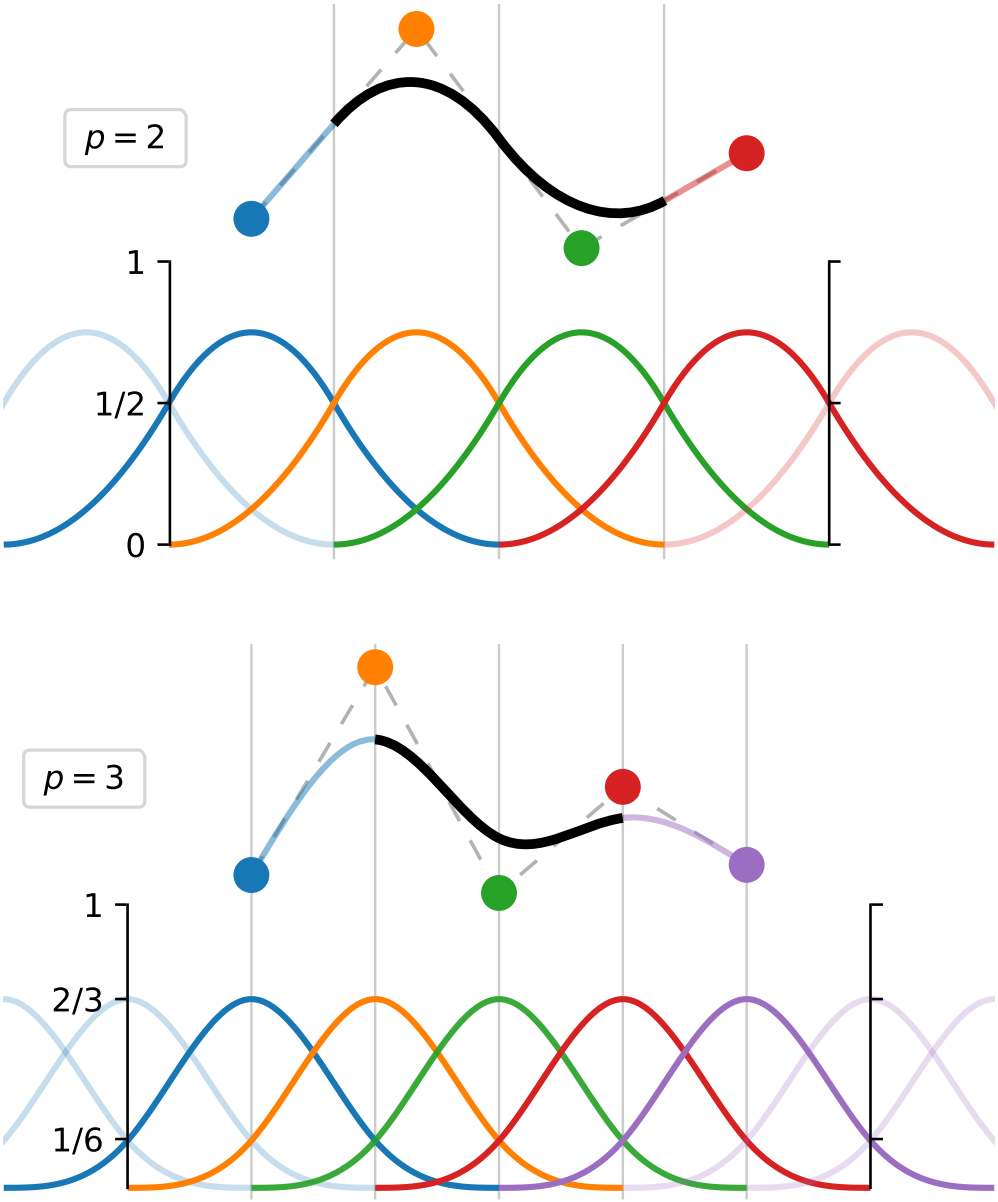


Figure 4.8: Examples of two open uniform B-splines and their basis functions below. Control points are spaced evenly along the x-axis such that knots (vertical lines) align with their associated segments. The top spline is quadratic (degree $p = 2$), while the bottom is cubic ($p = 3$). In both, the left- and rightmost basis functions go to zero at the middle knot, and so another curve that shares the middle p control points is guaranteed to overlap exactly at that knot. Note that the sum of basis functions must always add up to one, so to attach a curve to its endpoints, the final control point is repeated p times. The curves there are also slightly transparent, to match the corresponding repeated basis functions.

routing nodes are split twice to guarantee three shared control points). This appears to be the choice of the original authors (Bach et al., 2017) in their provided source code. The benefit of cubic splines is that they are C^2 continuous, i.e. have no sudden jumps in curvature, and having this extra smoothness is more aesthetically pleasing. Regardless, for all figures here except for the bottom of Figure 4.8 quadratic splines are used, along with the first method of joining to end points.

Note that drawing a spline for every edge is not the only way to render the graph, and also introduces a great deal of redundancy due to the large amount of overlapping portions. If a purely confluent drawing is desired, then it is not necessary to redraw overlapping segments. Allowing bundles to be relaxed (Bach et al., 2017, Fig. 18) is, however, a useful option that rendering each link does provide.

4.2.6 Directed graphs

The definition for a directed confluent drawing is almost the same as the undirected case, except with the caveat that any path can only 'flow' in one direction. This is analogous to preventing trains on tracks from crashing into each other. Formally, for a directed confluent drawing B, this extra condition (Dickerson et al., 2005) is:

- Locally monotone curves in B may share some overlapping portions, but the edges sharing the same portion of a track must all have the same direction along that portion.

This means that directed graphs lead to a slightly different power-to-routing graph conversion, where the description by Bach et al. (2017) says *"The only difference is that we create – as necessary – two junctions for each group, one for incoming and one for outgoing edges."* Here this short description will be elaborated upon, as

simply adding another routing node when its corresponding group has both incoming and outgoing edges is not enough to guarantee that all portions will only flow in one direction. This is because any power edges at a power group are propagated down the hierarchy, so that even if flows are correctly directed into separate junctions at one group, they may still clash further down the tree. Therefore, all descendants of any such group must also have two routing nodes: one for outgoing edges (flowing up the hierarchy) and one for incoming edges (flowing down the hierarchy).

The description of the improved power graph construction algorithm from Section 4.1.1 is also easily applied to directed graphs, by splitting neighbour sets into incoming and outgoing edges. This is how the original algorithm is described by Dwyer et al. (2014). In other words, the formulations for the heuristics in Equation (4.3) instead become

$$\kappa_{\cap}(m,n) = |N^+(m) \cap N^+(n)| + |N^-(m) \cap N^-(n)|, \tag{4.4}$$

$$\kappa_{\triangle}(m,n) = |N^+(m) \triangle N^+(n)| + |N^-(m) \triangle N^-(n)| \tag{4.5}$$

where N^+ is a set of outgoing edges, and N^- a set of incoming edges.

4.2.7 Planarity

Here the third condition in the confluent definition (Section 4.1.2) will be discussed, which requires planarity. Unfortunately, the method of Bach et al. (2017) does not offer any guarantee of planarity due to the use of a force-directed method to lay out the graph. While the authors do recognize this, and make the distinction that their drawings are 'non-planar confluent', the loss of this condition means that almost any drawing, with or without curved edges, satisfies a now-trivial definition. There also exist graphs that have been proven to not admit any confluent

drawing (Dickerson et al., 2005), and so it is impossible for any algorithm to produce confluent drawings for general graphs. However, the planarity condition can be relaxed in the context of finding a drawing that reduces the number of crossings, for example in layered confluent drawings (Eppstein, Goodrich, and Meng, 2007). Bach et al. (2017) do not explicitly address this question, but in practice the approach can greatly reduce the number of crossings, making it a practical method to enhance readability.

Furthermore, it is easy to see that a planar ARG can always lead to a planar drawing, as long as one is careful to avoid crossings at routing nodes (which is always possible because every routing node is simply a single junction). If it is assumed that there are no redundant routing edges without adjacencies routed through them, this means that the complete confluent definition can be satisfied if and only if the ARG is also planar. This naturally implies a new subset of confluent drawings, much like the Δ-confluent (Eppstein, Goodrich, and Meng, 2005) or strict confluent (Eppstein, Holten, et al., 2013) subclasses, which can be found only through a planar power-to-routing graph conversion. Such drawings will be named *power-confluent*. This new classification will now be proven to be a subclass of strict confluent drawing.

Stricter than strict

The definition of strict confluent drawing has only the additional restriction that there can be at most one smooth path between vertices, and there cannot be any paths from a vertex to itself (Eppstein, Holten, et al., 2013). For the power-confluent case, because the mapping of edges to power edges is surjective (Royer et al., 2008), each edge can only correspond to a single path through the power graph hierarchy, and therefore also through the routing graph behind the confluent drawing. Every power-confluent drawing is therefore also strict confluent.

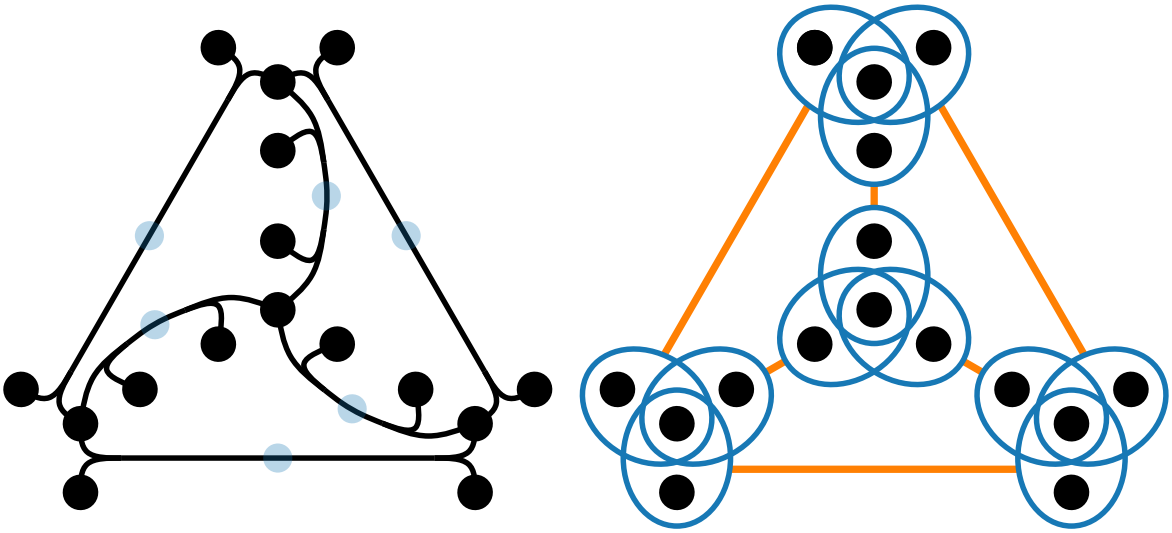


Figure 4.9: An example of a strict confluent drawing that cannot be reverse-engineered into a valid power graph, unless power group boundaries are allowed to overlap. On the left is the confluent drawing (with junctions marked by transparent blue circles), and on the right is a power graph that shows all possible groupings to match these junctions. The proof is further explained in Section 4.2.7.

The reverse is not true, because another condition of power graphs is that groups must be disjoint (Royer et al., 2008), i.e. their boundaries may not overlap. There exist confluent drawings that cannot result from power graphs unless this condition is broken, for example the tetrahedron-like structure in Figure 4.9. In this structure there are six junctions, and so at least six power groups are required in the corresponding power graph; however, there is a maximum of four that will not overlap, as there can only be one for each 'corner' of the 'tetrahedron'. This is due to the middle vertex of each 'corner' sharing junctions with all three surrounding it, which means that grouping any of the three valid pairings will block out the other two.

Note that the graph itself is not necessarily impossible to draw in a power-confluent manner, but that this particular drawing could never result from the algorithm presented in this paper. Power-confluent is therefore a strictly stronger condition than strict confluent.

4.3 Discussion

The work presented here has developed the ideas presented by Bach et al. (2017) by both improving the power graph decomposition step (Section 4.1.1) and fixing some practical issues with the conversion to a confluent drawing (Section 4.1.2). Source code for all the algorithms discussed can be found at `www.github.com/jxz12/pconfluent`.

Further directions could involve developing methods to find power-confluent drawings, by guiding the search algorithm towards solutions that produce planar routing graphs. Methods such as simulated annealing or Monte Carlo may also prove useful for improving the greedy search presented here.

It is also not clear how often the theoretical issues raised in Sections 4.2.2 and 4.2.3 would occur in real world data. The pathological power graph example in Figure 4.5 is clearly a rare situation, and it is further not known if the power graph construction algorithm outlined in Section 4.1.1 and improved in 4.2.1 would ever produce such a power graph. Future work could include analysing a dataset of graphs and searching for whether/how often such a situation ever occurs.

On the theoretical side, it may be the case that relaxing the non-overlapping group boundary condition, as explored by Ahnert (2014), could result in an equivalent classification to strict confluent drawing. The potential proof or refutation of this equivalence is left as an open question, along with determining the exact relationship between power decomposition and confluent drawing in general.

To finish on a practical note, a more tailored layout algorithm than standard force-directed methods will be necessary for the algorithm to become a truly practical tool, as layouts can often become tangled and unreadable. See Figure 4.10 for two examples of this on graphs previously studied in this chapter. The layout function

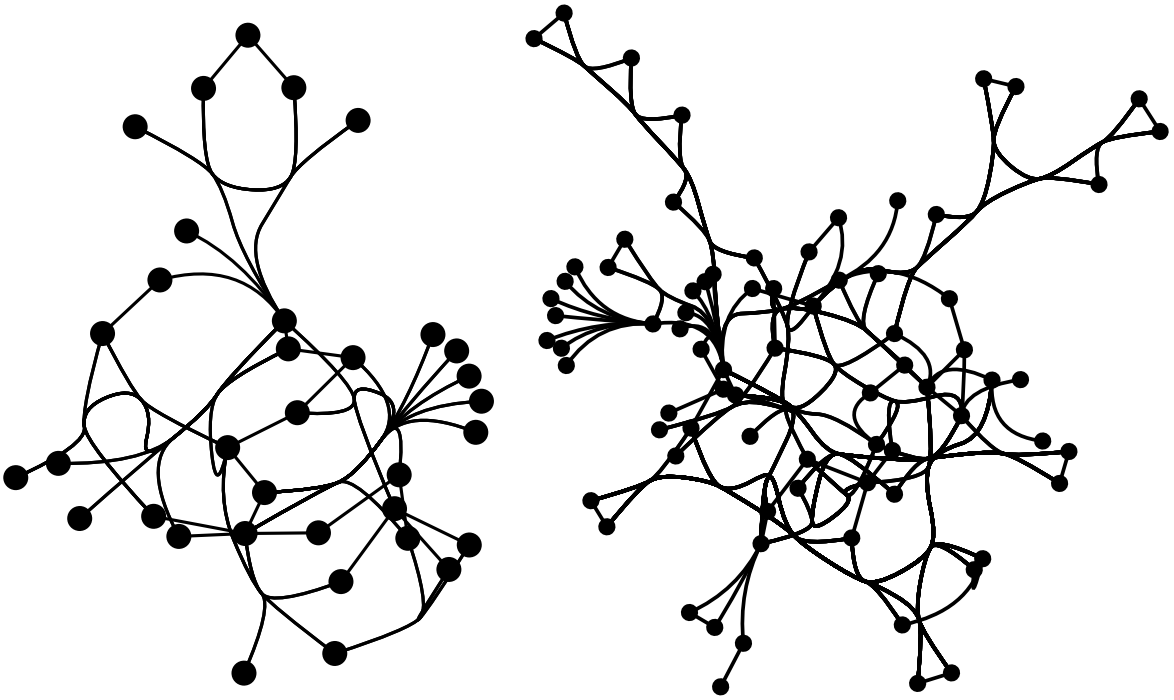


Figure 4.10: Two examples of graphs drawn with Algorithms 3 then 4, in Figures 4.2 and 4.7 respectively. On the left is `karate`, previously shown in Figure 3.7, top. On the right is `lesmis`, previously shown in Figure 3.3.

(Figure 4.7, line 10) is currently only given the routing graph as input, but may benefit from extra information, such as which edges are power edges and which are hierarchical. An effective radial layout that can avoid crossings, like the one shown in Figure 4.6 but automatic, may also offer a superior solution.

Chapter 5

EcoBuilder

The preceding chapters focused on the algorithms and methods underlying visualisation techniques, but it is important to remember that visualisation is a discipline with practical applications firmly in mind as the end goal. The work in this chapter will explore such a goal by moving towards an engineering focus, specifically the design and development of *EcoBuilder*, a research-oriented video game about building ecosystems.

5.1 Background

The unique selling point of EcoBuilder is that its ecosystem simulation is based on the same mathematical models studied by theoretical ecologists. The review of background literature in this section will therefore begin by discussing the models chosen in the following Section 5.1.1. With the knowledge of *how* pairs of species interact with each other, the following Section 5.1.3 will study the choice of *which* species interact at all. Section 5.1.4 then explores prior work behind how best to show the user a network whose topology changes over time, which is important as



Figure 5.1: Screenshots of EcoBuilder displayed on an iPhone and an iPad. The particular ecosystem shown depicts the bottom-right plant going extinct due to apparent competition, as well as the top-right animal going extinct due to competitive exclusion.

the player will be building this topology themself. Lastly, Section 5.1.5 will outline a brief history of previous attempts to crowdsource research through the citizen science approach, and give context to why EcoBuilder was created.

5.1.1 Predator–prey interactions

Ever since Isaac Newton formulated his physical laws of motion, it has become clear that nature speaks in the language of differential equations. From simple mechanical systems such as the oscillating swing of a pendulum (Fulcher and Davis, 1976), to the cosmic dance of celestial bodies in space (Marchal, 2012), and even the electrical action potentials across every neuron in the brain that is deciding which words should go into this sentence (Hodgkin and Huxley, 1952), calculus has time and again been an invaluable tool for describing the natural world.

The behaviour of ecosystems is no different, where the most widely used models function by defining the change in abundance (measured by biomass per unit area) over time. There are many such examples of this, ranging from the simple and linear systems to be studied here, to highly complex data-driven models such as *EcoPath with EcoSim*, which has been used for decades to inform environmental policy for aquatic conservation (Christensen and Walters, 2004). Other models include ones that consider the effects of temperature on metabolic rate (Savage et al., 2004; Dell, Pawar, and Savage, 2014), spatio-temporal dynamics such as turbulance in water (Watteaux, Stocker, and Taylor, 2015), or evolution to produce the correct dynamics as an emergent property (Laird, Lawson, and Jensen, 2008). Arguments have also been made for second-order differential equations to describe such dynamics (Colyvan and Ginzburg, 2003).

But perhaps the oldest and most commonly studied of these models is known as

the Lotka-Volterra equations, defined as

$$\frac{d\mathbf{z}_i}{dt} = \mathbf{z}_i(\mathbf{r}_i + \sum_j \mathbf{A}_{ij}\mathbf{z}_j) \tag{5.1}$$

where $\mathbf{z}_i$ is the abundance of species i, $\mathbf{r}_i$ is its growth rate (or death rate if it is negative), and $\mathbf{A}_{ij}$ is the strength of the interaction between two species, which is positive if i eats j and negative if j eats i. This can be written to include all species simultaneously as

$$\frac{d\mathbf{z}}{dt} = (\mathbf{r} + \mathbf{A}\mathbf{z})\mathbf{z}^T. \tag{5.2}$$

The matrix $\mathbf{A}$ is known as the *interaction matrix*, and is called that because it contains all the information on how each pair of species interact with each other. There are five classifications of interaction possible in such a model, designated by the nature of the values in the corresponding pair of values in this matrix. For example, if $\mathbf{A}_{ij} > 0$ and $\mathbf{A}_{ji} < 0$, then it is a standard example of species i preying on species j, where the 'sign' of the interaction is $(+,-)$. The remaining types are mutualism $(+,+)$, competition $(-,-)$, commensalism $(+,0)$, and amensalism $(-,0)$. Different models include differing amounts of each type of interaction, and the proportion of types can strongly influence the behaviour of the system (May, 1973; Allesina and Tang, 2012; Tang, Pawar, and Allesina, 2014). However the predator–prey type is the most commonly studied and most prevalent in nature, and so will be the only one considered here.

The purpose of the above equations is to use them to find values of $\mathbf{z}$, and a standard way of doing this for differential equations is to simulate by performing numerical integration at each time step, using methods such as Runge-Kutta (Press et al., 2007a).[1] However the simplicity of the model chosen allows something dif-

[1] This was used in older versions of the game, but brings with it numerical problems such as stiff equations leading to instability (Press et al., 2007a), or having to choose a suitable integration time step. It also adds the issue of having to choose an arbitrary abundance threshold for species to go extinct.

ferent: it can be analytically solved for an *equilibrium* point. This is the solution at which the interactions fully balance each other such that the instantaneous change in abundance is zero. Note the similarity of this method to the graph layout algorithm of Tutte shown in Section 2.1, Equation (2.1). This is done by setting the left-hand side of Equation (5.2) to zero, where it is clear that the system contains many trivial solutions with any species $\mathbf{z}_i = 0$, which corresponds to the natural case of an extinct species staying extinct. This also means that there are an exponential number of solutions, with each species at a zero or non-zero abundance. However, the strictly non-zero solution is the one of interest, which occurs at

$$\mathbf{z}^* = -\mathbf{A}^{-1}\mathbf{r}. \tag{5.3}$$

where $\mathbf{z}^*$ contains the abundances of every species at equilibrium. This is equivalent to solving a linear system of equations and so can be solved exactly by numerical methods. If the solution $\mathbf{z}^*$ contains any negative numbers the system of equations is said to not be *feasible* i.e. it is not possible for all species to coexist.

Local asymptotic stability

If the system is feasible, another benefit of this technique is that it allows for the derivation of the *local asymptotic stability* of this fixed point, which has been widely used in the literature as a mathematically elegant and computationally tractable measure of the stability of food webs (May, 1973; Emmerson and Yearsley, 2004). This definition will be henceforth referred to simply as stability. To find if a system is stable, the first step is to find the Jacobian matrix

$$\mathbb{J} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{z}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{z}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial \mathbf{z}_1} & \cdots & \frac{\partial f_n}{\partial \mathbf{z}_n} \end{bmatrix} \tag{5.4}$$

where f_i is the right side of Equation (5.1) and $n = |V|$ is the total number of species (vertices) in the food web (graph). This is the matrix of all possible partial derivatives of a system, which describes the instantaneous behaviour of the system by linearising it. This Jacobian is then evaluated at the feasible equilibrium point $\mathbf{z}^*$, resulting in what is known as a *community matrix*. For the Lotka-Volterra equations studied here, this is

$$\mathbb{J}|_{\mathbf{z}^*} = \begin{bmatrix} \mathbf{r}_1 + 2\mathbf{A}_{11}\mathbf{z}_1^* + \sum_{i\neq 1}^{n} \mathbf{A}_{1i}\mathbf{z}_i^* & \cdots & \mathbf{A}_{1n}\mathbf{z}_1^* \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{n1}\mathbf{z}_n^* & \cdots & \mathbf{r}_n + 2\mathbf{A}_{nn}\mathbf{z}_n^* + \sum_{i\neq n}^{n} \mathbf{A}_{ni}\mathbf{z}_i^* \end{bmatrix} \tag{5.5}$$

which also conveniently simplifies to $\mathbb{J}|_{\mathbf{z}^*} = \mathrm{diag}(\mathbf{z}^*)\mathbf{A}$, after substituting $\mathbf{z}^*$ back in using Cramer's rule (May, 1973). The system can then be declared locally stable if the largest real part of any eigenvalue of the community matrix is non-negative

$$\Lambda \leq 0. \tag{5.6}$$

An intuitive physical interpretation of this type of stability can be seen by imagining the two ways of holding a chopstick vertically when gripping only a single end. If the chopstick is pointing upwards, even the slightest nudge will make it fall over due to gravity. If it is dangling downwards, gravity will reset its position after a nudge, and it can be called stable. In the context of ecosystems, the chopstick is a species, and the nudge is an environmental perturbation such as the loss of individuals of one or more species' populations due to as natural disaster, or the addition of individuals to one or more species' populations due to immigraton.

The behaviour of the community matrix $\mathbb{J}|_{\mathbf{z}^*}$ is what is commonly studied in mathematical ecology, due to the fact that it can be reverse-engineered to apply to almost any system of differential equations. It therefore sidesteps the problem of choosing which model to use, by jumping straight to this matrix. It is also the

source of the long standing 'complexity vs. stability' debate in the world of ecology, because it can be shown that as the number of species, i.e. size of $\mathbf{A}$, grows, the chance of Equation (5.6) being satisfied tends to zero, even for small ecosystems of only a few dozen species (May, 1973). The reason for the debate is that systems exist in the real world with many more species than the math suggests is possible, and many subsequent works have studied what features of ecosystems allow such a contradiction to occur. For example, Tang, Pawar, and Allesina (2014) show that the correlation between pairs of elements in $\mathbf{A}$ can improve stability, Brose, Williams, and Martinez (2006) study the consequences of body mass, and Johnson, Domínguez-García, et al. (2014) study the layered nature of groups of species.

However a common criticism is that this method assumes the existence of a feasible equilibrium in the first place, and this is far from guaranteed. It has also been shown that in simple systems like the Lotka-Volterra equations studied here, that feasibility almost always leads to stability anyway (Dougoud et al., 2018). For this reason, the work done here will focus solely on feasibility.

5.1.2 Metabolic scaling

The steps so far have discussed how to find abundances given the interaction matrix $\mathbf{A}$, but it has not yet been discussed how to find the numerical values to go into the matrix in the first place, i.e. its parameterisation. To fill this gap, a set of constraints known as *metabolic scaling* will be used. It leverages the idea that each species has a metabolism rate that can be measured, i.e. the speed at which an organism converts a food source into either energy for movement or materials for growth.

Metabolism is useful for two main reasons. The first is that, as it essentially mea-

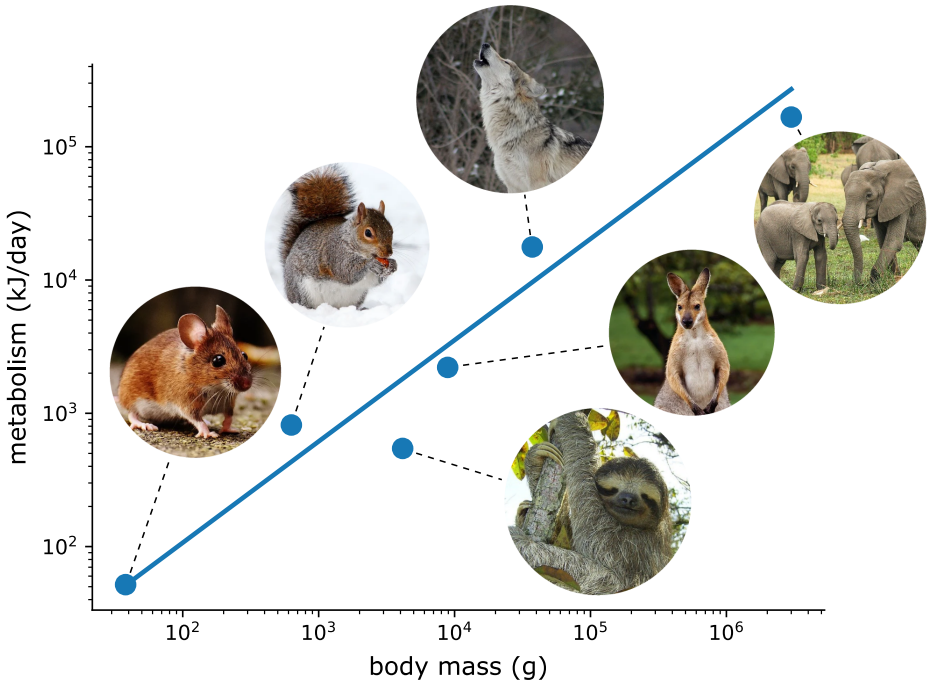


Figure 5.2: A plot of metabolism against individual body mass for a selection of animals. The slow-moving sloth is a good example showing that the model is imperfect, but still a very good estimate when considering the complexity of life. Data taken from (Nagy, Girard, and Brown, 1999), power law slope and intercept from Fig. 2 in Brown et al. (2004). All photos taken from `www.pixabay.com` under the Pixabay License.

sures the energy output of a species, it is strongly correlated with other traits such as movement speed (Hirt et al., 2017) and therefore has been successfully used for parameterisation of $\mathbf{A}$ (Tang, Pawar, and Allesina, 2014; Brose, Williams, and Martinez, 2006; Pawar, 2015). The second is that whole organism metabolic rate in multi-cellular eukaryotes scales positively with body mass. The study of the relationships between mass and other biological traits is known as *allometry*. Some examples of species and their metabolisms can be seen in Figure 5.2. Note that scaling is isometric to superlinear in unicellular eukaryotes and prokaryotes (DeLong et al., 2010), and that body temperature is also closely related to individual metabolism (Savage et al., 2004). These cases will however be left out of scope for the work done here and left as a possible future direction.

Specifically, the allometric functions used for the species-wide reproduction rates

$\mathbf{r}$ in Equation (5.1) are

$$\mathbf{r}_i = \begin{cases} b_0 \mathbf{m}_i^{\beta-1} & \text{if } i \text{ is producer} \\ -d_0 \mathbf{m}_i^{\beta-1} & \text{if } i \text{ is consumer} \end{cases} \tag{5.7}$$

where b_0 and d_0 are constants to normalise birth and death rates, and $\mathbf{m}_i$ is the body mass of species i in kilograms. The exponent β is an empirically verified relationship between metabolism and individual body mass (Brown et al., 2004), and it is taken minus one in order to convert it into mass-specific units. Remarkably, the same exponent of $\beta = 3/4$ can be applied to species at almost all sizes, from the smallest bacteria to the largest blue whale (Kleiber, 1947). It is theorised to take a value of $3/4$ – rather than the $2/3$ that one would expect from a spherical surface area to volume ratio – due to the fractal spreading pattern of tubes (such as veins) (West, Brown, and Enquist, 1997), although not without controversy (Agutter and Wheatley, 2004). Note that since $\beta = 3/4$, the exponent as a whole is negative, implying negative scaling of body mass to metabolic rate at the mass-specific level.

The strengths of predator–prey interactions are defined as

$$\mathbf{A}_{ij} = \begin{cases} -a_0 \mathcal{F}(i,j) \mathbf{m}_j^{-1} & \text{if } i \text{ is prey} \\ e_0 a_0 \mathcal{F}(j,i) \mathbf{m}_i^{-1} & \text{if } i \text{ is predator} \end{cases} \tag{5.8}$$

where a_0 is a normalising constant, e_0 measures biomass conversion efficiency, and $\mathcal{F}(i,j)$ is a function that captures the scaling of interaction rate in terms of consumer mass as well as consumer–resource body mass ratio. The $\mathbf{m}^{-1}$ factor at the end is to convert it into mass-specific units, just as the minus one in the exponent of Equation (5.7).

The function $\mathcal{F}(i,j)$ can be expressed in many ways, and is more difficult to measure than individual metabolism as it involves the interaction between two species.

To overcome this, Pawar, Dell, and Savage (2012) constructed a mathematical model that simulates species as randomly moving individuals, such that they can be treated as particles colliding under a Brownian motion approximation (Ocubo, 1980). Traits such as movement speed and mutual detection distance between consumer and resource have allometric relationships, and so were carefully combined (and validated against empirical data) to construct a mechanistic model of species interaction rate. The final resulting formulations for $\mathcal{F}(i,j)$ are

$$\mathcal{F}(i,j) = \begin{cases} \mathbf{m}_j^{p_v+2p_d}\sqrt{1+\left(\frac{\mathbf{m}_i}{\mathbf{m}_j}\right)^{2p_v}}\left(\frac{\mathbf{m}_i}{\mathbf{m}_j}\right)^{p_d} & \text{if active capture} \\ \mathbf{m}_j^{p_v+2p_d}\left(\frac{\mathbf{m}_i}{\mathbf{m}_j}\right)^{p_d} & \text{if grazing} \end{cases} \tag{5.9}$$

where p_v and p_d are allometric scaling exponents derived to capture species velocity and reaction distance, respectively. The top version for 'active capture' captures the case where both species move around, e.g. foxes hunting rabbits, and the bottom for 'grazing' captures only the consumer moving, e.g. deer feeding on grass. These are the functions that will be used for all work done here, where all primary producers will be assumed to be stationary, and all consumers are assumed to be mobile. The numerical values of the constants used for this model can be seen in Table 5.1.

The values of $\mathcal{F}(i,j)$ and therefore $\mathbf{A}_{ij}$ reduce to constants, and so the functional behaviour of these interactions is still linear. Linearity has been criticised in the past for being too simplistic, and more complex models such as Type II or III functional responses (Holling, 1973) have introduced non-linear functions into $\mathbf{A}$ in order to capture the effects of handling time and predator satiation, respectively. However, due to the added complexity to the model and the fact that the equilibrium calculation in Equation (5.3) would become much more difficult due to a non-linearity, it was decided that only a linear functional response will be consid-

Table 5.1: Constants used for the model in Section 5.1.1, taken from Pawar, Dell, and Savage (2012). The value for e_0 takes two values to account for prey being more difficult to digest as a plant (0.2) than an animal (0.5), close to the naturally observed values (Lindeman, 1942; Kozlovsky, 1968).

Parameter	Description	Value
b_0	Normalising constant for birth rate	1.71×10^{-6}
d_0	Normalising constant for death rate	4.15×10^{-8}
β	Scaling exponent for metabolism	3/4
a_0	Normalising constant for search rate	8.31×10^{-4}
p_v	Scaling exponent for velocity	0.26
p_d	Scaling exponent for detection distance	0.21
e_0	Biomass conversion efficiency	0.2 or 0.5

ered here.

The equations outlined above only require two parameters per species in order to parameterise almost all of Equation (5.1): the choice of whether the species is a producer, and the choice of its body mass. These are two intuitive and biologically meaningful traits for a player to consider. The values that require parameterisation are the diagonal values of $\mathbf{A}$, which will be discussed in the following section.

Intraspecific interference

It is well known that no species can grow exponentially forever, as there is always some point at which the abundance of any species reaches its carrying capacity. This applies to almost every example of exponential growth in the real world, from the growth of a business on the stock market, to the spread of a deadly virus.

This is captured by the diagonal elements of the interaction matrix $\mathbf{A}_{ii}$. This can be interpreted as the amount that species inhibits itself, or in other words the *intraspecific interference*. It is always a negative value, because otherwise the abundance of a species would grow even faster than exponentially, and can be inter-

preted as competition between individuals over space.

This value has a strong effect on the dynamics of the model. In fact, it can be shown that any ecosystem can be made stable by simply increasing the magnitude of the diagonal values of the community matrix in Equation (5.5). Barabás, Michalska-Smith, and Allesina (2017) further showed the importance of these values, as they proved that it only takes one weak interference value to qualitatively affect the stability of an ecosystem.

Unfortunately, there is no easy way to measure this value in real world ecosystems, as quantifying the amount a species competes with itself is difficult. Previous works have overcome this problem by assuming every species can coexist at an abundance following Damuth's law, which states that the species with larger body masses tend to have smaller populations (in terms of number of individuals). The minimal constant value for all $\mathbf{A}_{ii}$ is then calculated for these abundances (such that eigenvalues are pushed to the left and Equation (5.6) is satisfied), and used as a proxy for the health of an ecosystem (Tang, Pawar, and Allesina, 2014; Pawar, 2015). However this is the opposite order of causation, as the behaviour of $\mathbf{A}$ should be the mechanism underlying Damuth's law in the first place.

Without a good way of deriving these diagonal values mechanistically, the work here will simply leave them as another input parameter chosen by the player. This interference of each species, alongside the body mass to parameterise the off-diagonals as in the previous section, will therefore be the two degrees of freedom that determine the behaviour of each species.

5.1.3 Food web topology

The final missing piece from the model needed to fully simulate an ecosystem to decide which species should interact with each other in the first place. The study

of the topology of food webs has seen many models being developed to capture the non-randomness that exists in their structure.

One of the main features that such models attempt to incorporate is the observation that big things generally eat things smaller than themselves. An early version of this is the cascade model by Cohen, Briand, and Newman (2012), which begins by placing every species along an axis, and then only allows species to prey on other species that are lower than them on this axis. The placement on this axis can, but is not necessarily, interpreted as body mass. A notable improvement to this is the Niche model of Williams and Martinez (2000), who leveraged the fact that big things eat small things, but not too small. They therefore constrain species to only prey on other species within a certain range on the axis, as shown in Figure 5.3, top.

Other attempts include the nested hierarchy model (Cattin et al., 2004) which captures the clustered nature of food webs, the trophic coherence model (Johnson, Domínguez-García, et al., 2014) which captures the layered nature of groups of predators, or the speciation (Rossberg et al., 2006) and diet breadth (Petchey et al., 2008) models, which simulate evolutionary and foraging processes, respectively, to produce structure as an emergent property.

However, none of these models can capture the topology of food webs fully satisfactorily. Furthermore, thinking about the options to give a player who is building a food web, it would be much more interesting to have full control over which interactions are realised. That is the choice used from this point on, but a good question to ask at this point is whether it is possible for humans to comprehend the topology of real world webs in the first place.

Dimensionality

One thing that all the aforementioned models share is *low dimensionality*. This essentially means that there exists a structural pattern that can be largely explained using rules and models without too many parameters. It is important that the dimensionality of topologies players are tasked with recreating is low, because humans struggle with understanding high-dimensional information. After all, the algorithms studied in previous chapters all aimed at reducing the dimensionality of graphs down to two dimensions. To gain an idea of just how complicated structures can become, the number of possible edges in a general directed graph is $n(n-1)$, which means that there are $2^{n(n-1)}$ possible configurations for the user to try, assuming edges can go in both directions. Even if edges can only in one direction then there are $3^{n(n-1)/2}$, which is still worse than exponential. Note that these calculations are for completely general graphs, and that there are extra constraints in food webs such as plants not being predators; the idea of just how complex these topologies can be nevertheless still holds.

It may then seem impossible for the player to effectively explore this search space, but low-dimensional topologies should mean that simple strategies and patterns can be used to successfully construct them. Fortunately, a study by Eklöf et al. (2013) collected a large amount of data to directly ask the question: what dimensionality do empirical food webs exhibit? Dimensionality in this case is defined as the number of *niche axes* required to capture the topology of a food web, which follows the same definition as the aforementioned niche model. An example of this is shown in Figure 5.3.

The question was answered by searching for the minimum number of niche axes required to describe the topologies present in the data, by applying an algorithm that searches the space by swapping certain species one by one, and then estimat-

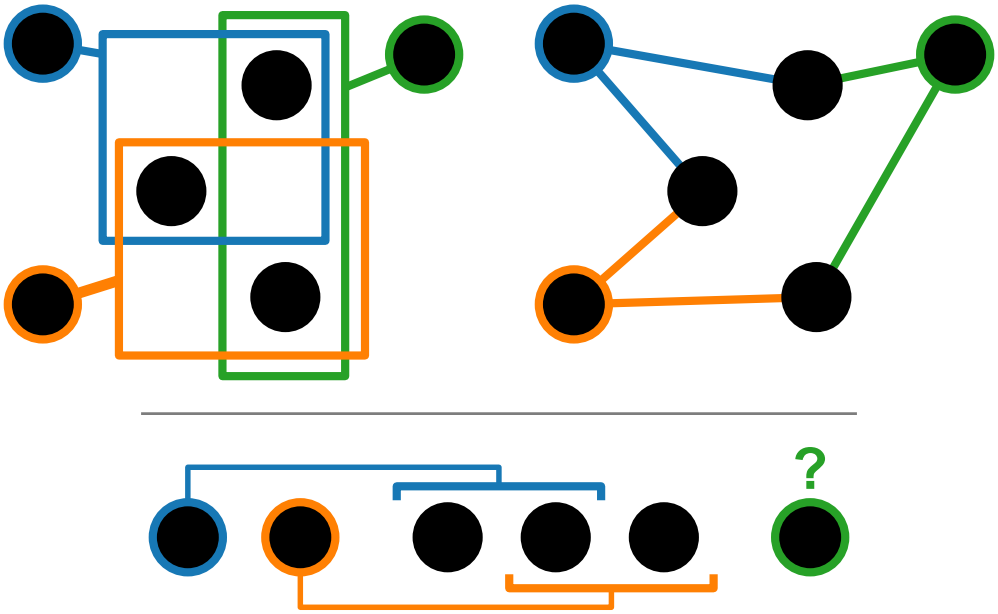


Figure 5.3: An example of a food web that cannot be one-dimensional, when following the definition in Section 5.1.3. Top-left is a two-dimensional niche drawing, where coloured boxes indicate the group of prey for a predator. Top-right is the corresponding node-link diagram. The bottom drawing shows why only having one dimension cannot position the prey such that each pair is adjacent.

ing the 'best' dimension using the Akaike information criterion (Eklöf et al., 2013). This is necessary because there does not exist a polynomial time algorithm for determining the number of axis required. Their result was that the number of dimensions needed to describe even the largest webs is surprisingly low. For smaller webs (fewer than 250 interactions) only one dimension was needed, and only four were needed for their best prediction on webs with thousands of species (with a mean of 1.395 over all webs).

This definition of dimensionality should not be confused with the widely-used interpretation of physical attributes that lie on an axis, such as body mass as described is Section 5.1.2. It is a strictly mathematical definition used to describe graph topology. This means that dimensionality is not a direct component of the game itself, but it does mean that relatively simple strategies and patterns should be able to describe any real food web topology, and therefore that humans can hope to form such strategies in order to play the game well.

Trophic levels

A final topological feature that will be considered is the *trophic level* of each species, specifically the prey-averaged trophic level (Williams and Martinez, 2004), defined as

$$\mathbf{y}_j = 1 + \sum_i \mathbf{P}_{ji}\mathbf{y}_i \tag{5.10}$$

where $\mathbf{P}_{ji}$ is the proportion of the diet of j that i contributes to. In other words, the trophic level of a species is exactly equal to the mean trophic level of its prey, plus one. The similarity of this definition to Tutte's algorithm back in Chapter 2, Equation (2.1) is worth noting, as the matrix to be solved here is also a graph Laplacian, defined as

$$\begin{bmatrix} 1 & -\mathbf{P}_{1,2} & \cdots & -\mathbf{P}_{1,n} \\ -\mathbf{P}_{2,1} & 1 & \cdots & -\mathbf{P}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ -\mathbf{P}_{n,1} & -\mathbf{P}_{n,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{5.11}$$

where most of the values of $\mathbf{P}_{ji}$ will be zero because the structure will be reasonably sparse. The matrix also has the nice property of being diagonally dominant, which means that iterative methods such as Jacobi or Gauss-Seidel are guaranteed to converge (Young, 2014). This will be made use of later in Section 5.4.1.

It is widely accepted that the trophic level of species has a useful meaning within the context of ecosystems (Post, 2002; Johnson, Domínguez-García, et al., 2014). This is because species with high trophic levels tend to be more biologically 'complex', as they are at the top of the food chain, like humans for example. Trophic level will therefore be made use of in Section 5.4.1 within a visualisation context.

A summary statistic based on the trophic levels of all species is known as *trophic coherence*, and is a measure of how homogeneous the layers of trophic levels are. For example, a perfectly coherent topology would have all integer trophic levels.

Specifically, coherence is defined as

$$\text{coherence}(\mathbf{y}) = \sqrt{-1 + \frac{1}{|E|} \sum_{\{i,j\} \in E} (\mathbf{y}_i - \mathbf{y}_j)^2} \tag{5.12}$$

and can be interpreted as the standard deviation of differences in trophic level over each edge. It is strictly larger than zero, where zero indicated perfect coherence, and so the larger it is the more incoherent the web. This quantity was studied by Johnson, Domínguez-García, et al. (2014), where it was shown that more coherent topologies result in more stable ecosystems. It will be used here to analyse the strategies taken by players to tackle challenges in the game.

5.1.4 Dynamic graph layout

The layout of the nodes shown to the player leverages the work done in Chapter 2 by minimising stress. However the topology of their network will change every time the player adds/removes a node/link, and so the layout must be recomputed each time an action is performed. A problem with this is that there is no guarantee that nodes will not be moved vastly apart in successive layouts, especially since stress is invariant to rotation, translation, and reflection. Another problem is that nodes with identical shortest paths to all other nodes in the topology may also swap positions, with no effect on the resulting stress. For example, the upper two nodes in the X shape in Figure 5.1 are topologically identical, and so swapping their positions would have no impact on the resultant stress. Simply re-performing the layout process every time a link is added results in these mismatches becoming a common occurrence.

This problem of aligning layouts between different topologies is known as *dynamic* graph layout in the literature, where the task is defined as preserving the *layout stability* of the drawing or the *mental map* of the viewer. Among the first

works to describe this formally are Eades, Lai, et al. (1991) and Misue et al. (1995), where different definitions of stability are presented, such as reducing the movement of the same node across successive layouts, and/or maintaining similar neighbourhoods of nodes in close proximity.

Archambault and Purchase (2012), Archambault and Purchase (2013), and Archambault and Purchase (2016) later resolved the real benefit of maintaining stability, as prior studies had surprisingly not found improvement in task performance when users were presented with more stable layouts. For example, previous tests gave graph readability tasks such as finding the node with the highest degree with no significant performance benefit Saffrey and Purchase, 2008. It was shown that that the real benefit of stability was to keep track of specific nodes and edges in an evolving graph, resulting in increased performance in *orientation* tasks, such as tracking specific locations or paths in the topology.

A naive method of maintaining layout stability is to use the previous layout as the initialisation for the next. This is effective at maintaining nodes close to each other, but unfortunately can throw the optimisation directly into a local minimum, leading to low-quality layouts. The work in Chapter 2 showed that stochastic gradient descent (SGD) does not require smart initialisation to effectively minimise stress, but unfortunately it is not good enough to escape from a bad (i.e. worse than random) initialisation. This is especially true when a single axis is constrained, as will be described in Section 5.2.3. Another common strategy is to simply pin nodes from previous layouts and only move ones that have directly changed (Ghani, Elmqvist, and Yi, 2012), but over the course of time this may lead to accumulated stress and tangled layouts.

Common methods of better solving the problem include *anchoring* and *linking*, which both involve keeping information from previous layouts in consideration. Anchoring is the idea of precomputing an ideal position for each node and adding

an attractive force for that node to its ideal position, although this is usually only applicable in offline situations where the sequence of topologies is already known. Linking applies a force between corresponding nodes in adjacent layouts, and so can be done in an online context. Brandes and Mader (2011) explored multiple ways of performing both in the context of stress minimisation, and found that the linking strategy worked best in most contexts for maintaining stability. They also quantified the trade-off between readability and stability, as one must give up some reduction in stress to keep nodes together across successive layouts.

A variety of other creative ideas to deal with this problem have also been proposed, such as tweaking a force-directed model to minimize Bayesian probability differences between layouts (Brandes and Wagner, 1997). Simplifying the union of the two graphs using a hierarchical decomposition, similarly to those studied in Chapter 4, and depicting only the movement of the hierarchy was also explored by Archambault (2009). Presenting the user with multiple timeslices of the transition in a comic-style grid, rather than through animation, has also been shown to improve task response times, although at the expense of accuracy (Archambault, Purchase, and Pinaud, 2011). A example of a modern method is Dynamic Network Plaid (Lee, Archambault, and Nacenta, 2019) which presents such timeslices on a large touch-screen display, opening up new realms of possibility in terms of interactive exploration of the data.

In the context of EcoBuilder, since the considered graphs are small i.e. fewer than a couple dozen vertices, it was decided that the state-of-the-art methods mentioned above were not crucial for the intended use case. However this would a top priority for any future work. The layout is simply recomputed on each topological change, and a linear equation known as the *Procrustes*[2] statistic was used to align

[2]The name Procrustes is derived from Greek mythology, and translates to 'stretcher'. It is based on the story of a man named Damastes, who was given it as a nickname because he would offer unsuspecting travellers a place to stay for the night, but proceed to stretch their limbs on a rack if they did not exactly fit the bed. Karma eventually caught up to Damastes when he experienced the

successive layouts. This simply measures the sum of squared distances of movements, and can be directly minimised using linear methods. Therefore, the differences in rotation, translation, and reflection layouts are neutralised, but other stability metrics are not considered and left as future work. Note that this statistic has been previously used to assess the similarity between layout algorithms (Ortmann, Klimenta, and Brandes, 2017). More specific details on implementation are described in Section 5.4.1.

5.1.5 Citizen science

At the time of writing, the video game industry has grown large enough to eclipse the film and music industries, combined (Egenfeldt-Nielsen, Smith, and Tosca, 2019). Its recent growth is due to the rise in mobile and competitive gaming, which have rounded out the market to appeal to casual and hardcore audiences respectively. Their popularity has also spilled into the world of research, with a variety of research-oriented games being produced in a range of disciplines. The motivation behind making such games is twofold, as they can simultaneously perform outreach through public engagement, as well as produce research outcomes through crowdsourcing of human intelligence through player gameplay.

This crowdsourcing is known as *citizen science*, and is not limited to the digital world; it has been successfully applied to physical projects such as measuring soil composition (Rossiter et al., 2015), tracking the global impact of light pollution (Cui, Shen, et al., 2020), or centralising the work of birdwatching enthusiasts to better track the population trajectories of bird species (Link, Sauer, and Niven, 2008). Even in the digital space, citizen science does not necessarily require a gamified context in order to perform the crowdsourcing. One of the oldest dig-

same fate as his guests at the hands of Theseus (Cox and Cox, 2000).

ital citizen science projects is *Galaxy Zoo* (Masters and Galaxy Zoo Team, 2019), which tasks players at identifying different types of galaxies from telescope data, in order to build a map of the universe. They have successfully classified the morphologies of galaxies for over a decade, and the data collected has even been used in machine learning contexts (Walmsley et al., 2020).

The projects most related to the work done here are however the gamified ones. Some of the most successful examples of this include:

- *Foldit* (Cooper et al., 2010), a game where the player is given the ablity to fold proteins, with the objective being to find folded configurations with the lowest *Rosetta* energy. This is important because real proteins fold into low energy configurations, and knowing the topology of such states is important to understanding their chemistry. The search space of topologies is too large for computers to currently handle, and so they turn to the spacial reasoning of humans (although Google have recently made attempts at solving this with machine learning (Senior et al., 2020)).

- *Eyewire* (Bae et al., 2018), where players are tasked with analysing pictures of 3D images of brains, collected using an electron microscope. The game aims to leverage the pattern recognition abilities of humans in order to map the exact locations of huge numbers of neurons.

- *EteRNA* (Lee, Kladwang, et al., 2014), a game conceptually very similar to Foldit, except that it is applied to RNA sequences instead. It is also presented as a 2D interface instead of 3D, which presents a slightly different type of reasoning required to solve its puzzles.

Popular mainstream games have also included citizen science elements within their games. *Borderlands 3*, a game that has sold millions of copies, included a

block puzzle similar to Tetris that tasks the player with mapping DNA across gut biomes.[3]

The success of these projects shows that the general public has great potential as a scientific resource to be tapped into. People from non-scientific backgrounds are in general keen to contribute to a scientific cause; Ponti et al. (2018) discuss player motivations and came to the conclusion that participants are driven by scientific ideals such as collaborative progress and democratisation of knowledge. They also warn, however, that introducing too much of a competitive element to the gamification may produce cognitive tension between the two goals of collaboration and rivalry between players.

Canine inspiration

The story behind EcoBuilder is based around historical events in Yellowstone Park, USA. In 1926, wolves were declared officially extinct in the park due to an excess of hunting due to lack of regulation at the time. The consequence of this was that elk no longer had any natural predators, leaving them to feed excessively on the park's plants without risk of predation. The resulting supression on plant population had a knock-on effect to smaller mammals such as beavers and fish, causing their populations to struggle, and the health of the entire ecosystem to deteriorate. This continued until 1995, when ecologists decided to take the action of reintroducing wolves to the park, by capturing fourteen wolves from Canada and transporting them across the border (Smith, Peterson, and Houston, 2003).

Thankfully, the operation was a success due to the ensuing *trophic cascade:* elk once again had a natural predator, leading to the restoration of plants, bringing back beavers who could again build dams, drawing fish back into rivers, and so on

[3] See `www.youtube.com/watch?v=L_mH6Ak_Ny0` for their official promotional explanation.

and so forth (Dobson, 2014). EcoBuilder aims to give ordinary people the ability to make similar decisions in their own simulated ecosystems. These simulations will be performed using the equations in Section 5.1.1, and so phenomena like the chain reactions seen in Yellowstone can be recreated, as emergent properties of the underlying dynamical system.

The goal, in terms of citizen science, is to create a symbiotic relationship between players and researchers within the context of the game. Players will learn about how ecosystems function, and why they may fall apart given the wrong structural decisions. In return, players will be tasked with building ecosystems that offer potential solutions to the stability vs. complexity debate. A further visualisation experiment is also embedded within the game, with the aim of exploring different types of node layout for food webs. The design of the game such that these research outcomes of the game will be outlined in the following Section 5.2. A more detailed description of implementation and design details can be found in Section 5.4, which serves as an appendix to this chapter.

5.2 Experimental design

The gameplay of EcoBuilder is split into two sets of levels. The first set is called 'Learning World', and the second is 'Research World'. When the player first opens the game, they are greeted with two large buttons that lead to the two Worlds. Initially, Research World is locked and may only be entered when they complete all the levels in Learning World, as shown in Figure 5.4, left. This gives the player an incentive to complete the Learning levels, and also ensures that all players have received the same introduction before tackling the more difficult problems presented in the Research levels.



Figure 5.4: Screenshots to illustrate Learning and Research World. From left to right: the splash screen the user sees when they start the game, the list of levels in Learning World, and the leaderboards in Research World. A more detailed outline of the user interface within each level can be found in Section 5.4.

5.2.1 Learning World

Learning World contains 18 levels in total, all of which contain relatively simple scenarios designed to be instructive. For example, levels 3 and 4 both force the use of a heavy animal that cannot be saved with a single plant, to teach the player that heavier species consume slower (on a mass-specific basis) and so require more abundant food sources. Levels 5 and 6 present the player with a situation known as *competitive exclusion*, a well-known natural phenomenon that states that if two predators share the exact same prey, then the predator with a faster intake rate will almost always drive the slower to extinction.

The corresponding real-world example told to the player is the exclusion of red squirrels after the introduction of grey squirrels in Britain, due to their superior foraging ability (Gurnell et al., 2004). This principle, although predicted by the mathematical model, is however not as common in nature as one would think. Ex-

amples such as the 'paradox of the plankton' seemingly ignore the principle entirely, as tens or hundreds of plankton species are able to coexist despite sharing the same resources (Hutchinson, 1961); results from the game may help to reveal why this occurs.

To complete each level in the game, the player needs only to construct an ecosystem where all species can coexist. Each level additionally has the option to earn extra 'stars' by also maximising their score, thereby giving the player a motivation to better understand the underlying dynamics of the simulation. A description of the scoring metric used in the game can be found in Equation 5.17 in Section 5.4.2. Unfortunately, this metric does not have any direct significance in terms of answering a research problem, so is not suitable for crowdsourcing the answer to an ecological question. This issue will be discussed in the following Section 5.2.2, where the player will be given two alternate scoring metrics that are of greater research interest.

5.2.2 Research World

Research world contains 3 further levels for the player to play, each with a different scoring metric. These levels act more like a *sandbox*, where players may add up to 48 species (limited due to the uniqueness constraint explained below) in whatever topology they wish. They are given a single task: get as high a score as possible. This score in Level 1 is simple the same as in Learning World, i.e. Equation 5.17. Level 2 adds an extra one million points per each additional animal, which is so large that it effectively means that Equation 5.17 is only included as a tie-breaker. Level 3 also gives blocks of a million points, but for each additional max chain length instead of each animal. To incentivise players to reach for high scores, each of these three levels has a leaderboard for skilled players to aim to be

the best, as shown in Figure 5.4, right. The purpose of Research World is to see how players approach these three situations, all of which do not have an immediately obvious solution.

There are a couple of extra details that first must be addressed, however, before letting players loose. The first issue is that there are many trivial solutions if the player is allowed to add many identical species. This is because the main cause of extinctions in the simulation is due to chain reactions such as the aforementioned competitive exclusion, but these do not occur if species are identical (Armstrong and McGehee, 1980). For this reason, no two species are allowed to have the same set of traits. Since there are 9 possible values for body mass and 5 for interference, this results in an upper limit of 45 combinations, plus the binary choice of plant or animal. The number of plants is limited to 3, leaving a total of 48 species per web: more than enough to comfortably fill the screen of a mobile device.

5.2.3 Objectives

EcoBuilder is a *puzzle* game – the player is not tasked with time constraints or tests of dexterity. It is therefore a direct test of the ability of the player to understand the dynamics of the underlying simulation. This presents the opportunity to perform a controlled test to measure the extent of this understanding under different conditions.

In this case, two node layout methods are tested, where one is computed by simply minimizing stress, and the other additionally has the y-axis positions of species constrained to their trophic level. The layout with only stress optimised will be referred to as the *standard* layout, and with y-axis constrained as the *trophic* layout. See Figure 5.5 for an illustrative example of both layouts side by side. Further details on the implementation of these layouts can be found in Section 5.4.1

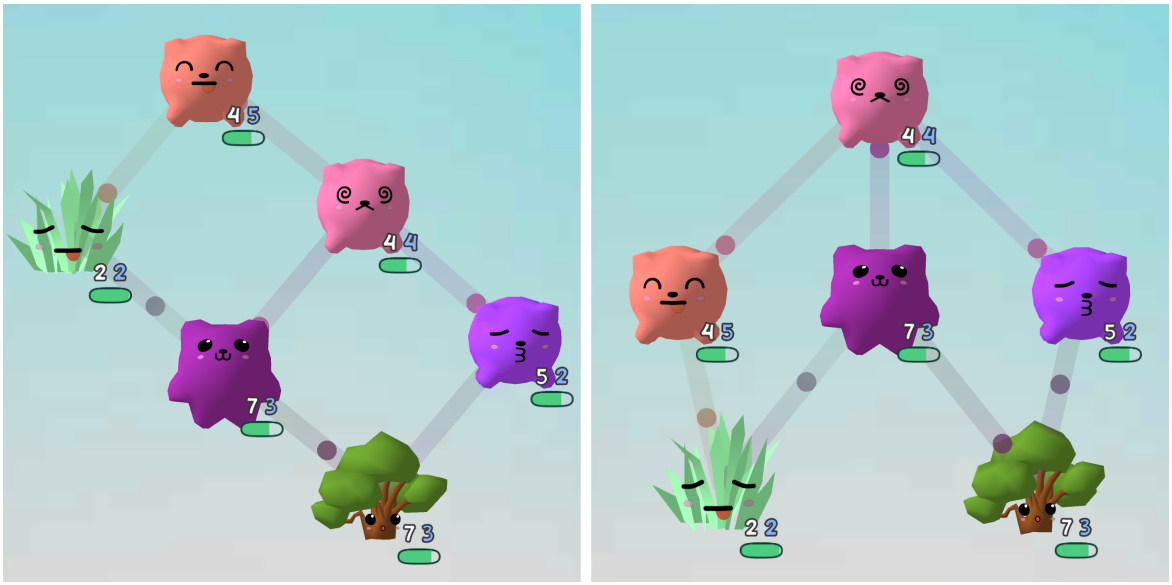


Figure 5.5: A side-by-side comparison of the same food web with and without trophic level constraints. The unconstrained layout is on the left, and the constrained is on the right. In the game itself, the little circles along links move to show the direction of biomass flow.

Other experiments in the literature generally give participants topological tasks such as finding paths between nodes (Bach et al., 2017; Okoe, Jianu, and Kobourov, 2018) or deciding which nodes would disconnect the network if removed (Purchase, 1997). A novel difference with the experiment here is that there is an actual dynamical system for the participant to further understand.

The fact that puzzle games are solved using human intelligence also brings the possibility of leveraging this intelligence to crowdsource research. As previously described in Section 5.1.3, the number of possible topologies in a directed graph without bidirectional edges is $3^{n(n-1)/2}$. To place this in perspective, with just six species this already corresponds to over ten million possible topologies. With twenty species, this is 4.5×10^{90}: more than the estimated number of atoms in the observable universe (Guth, 2003). Tasking the player with finding the patterns and strategies to navigate such complexity is the purpose of Research World.

With these details in mind, it is possible to state two objectives:

- **Objective 1:** to determine whether trophic level layout constraints will help players to complete Learning World faster.
- **Objective 2:** to task players with finding strategies to achieve high scores in Research World, and to explore whether these strategies reflect patterns found in reality.

The reasoning behind the first objective is because constraining species to their trophic level tends to orient edges upwards, as illustrated in Figure 5.5, which should help to preserve the player's mental map of the flow of biomass through the system. However this does remove a degree of freedom from the optimisation of stress, and so layouts may become more difficult to read for more dense topologies, thus hindering the player instead.

The second objective is the citizen science component of the game, where it is hoped that players will learn enough about the dynamical behaviour of ecosystems through their experience in Learning World, to solve the extra problems where the solutions are both unknown and involve many more species. The strategies taken by the top players will be analysed to see if real-world features materialise as emergent properties of the constructed ecosystems.

5.3 Results and analysis

Data from a total of the 842 players who completed at least one level, and over 9000 individual playthroughs were analysed to produce the results presented here. The analysis will begin with examining player performance in Learning World, and then finish with Research World.

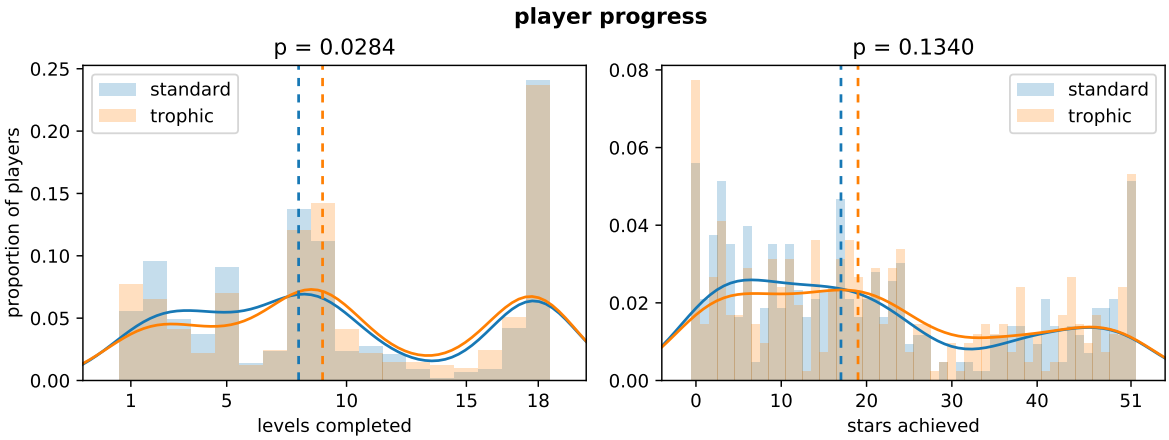


Figure 5.6: Bar charts showing how far players progressed through the game, for both the standard stress layout and trophically constrained groups. On the left is how many levels were completed, and on the right is how many stars were collected. The smooth curves denote a kernel density function over all bins, and the dotted vertical lines mark the median. The p-value for the null hypothesis of the medians values of both distributions being the same is written above each plot, calculated by generating 100,000 permutations of the test set.

5.3.1 Objective 1

Objective 1 concerns the performance of players as they progressed through Learning World, depending on the type of visualisation they were presented with, as detailed in Section 5.2.1. The first question to ask is: how many levels did players complete before stopping? This is answered in Figure 5.6, where bar charts containing the furthest level reached and number of stars collected can be seen. The test statistic studied here for both plots is the *median*, marked by vertical dotted lines for both plots, where it can be seen that progress is slightly further for the trophic group for both measures. To validate the significance of this result, a permutation test was performed with the null hypothesis that there is no difference in medians between the two layouts. This was done through a permutation one-tailed test (Good, 2006), by counting the number of times that the observed increase in median for the trophic group occurred over 100,000 permutations of the concatenated test set. The resulting p-values are written on top of the corresponding charts in Figure 5.6.

The p-value of 0.028 for levels completed shows a statistically significant result for which group made it further through the game, as defined a p-value below 5%. The corresponding result for number of stars achieved is not as convincing, but is still positive nonetheless. Looking at the shape of levels completed, there is a big cutoff at levels 8 and 9, which is a sign that those two levels where designed to be a little too difficult. However, almost all players with the perseverance to make it over that hump continued on to complete the remaining levels. This was the biggest group of players in general, coming in at just under 25% of all players.

However, this difference in progress through levels makes it incorrect to directly evaluate objective 1 by simply looking at the total play time of players, since playing more levels will increase total play time. A look at the data indeed confirms that the median play time of trophic players was longer. Because of this, the time element is better analysed on a level-by-level basis. This can be seen in Figure 5.7, where a histogram of the time taken by both sets of players is plotted for each of the 18 levels in Learning World, including p-values over the median for each, as in Figure 5.6.

Not many of the resulting medians are statistically significant, however almost every level has a better median for trophic than standard, with many of the more difficult later levels falling below a 10% threshold. Level 7 is the only level where the opposite case occurs, and the standard group performed consistently better. However it is also the level that players had the least trouble with in general, with the fastest completion time over all levels.

A peek into research world confirms the need for care when picking a visualisation, as the standard group achieved higher median scores than trophic on the first Research Level, with a p-value of 0.096 on the null hypothesis that the median scores from both groups is the same. This first level simply reuses the score metric in Equation (5.17) which takes connectance into account, and so food webs are

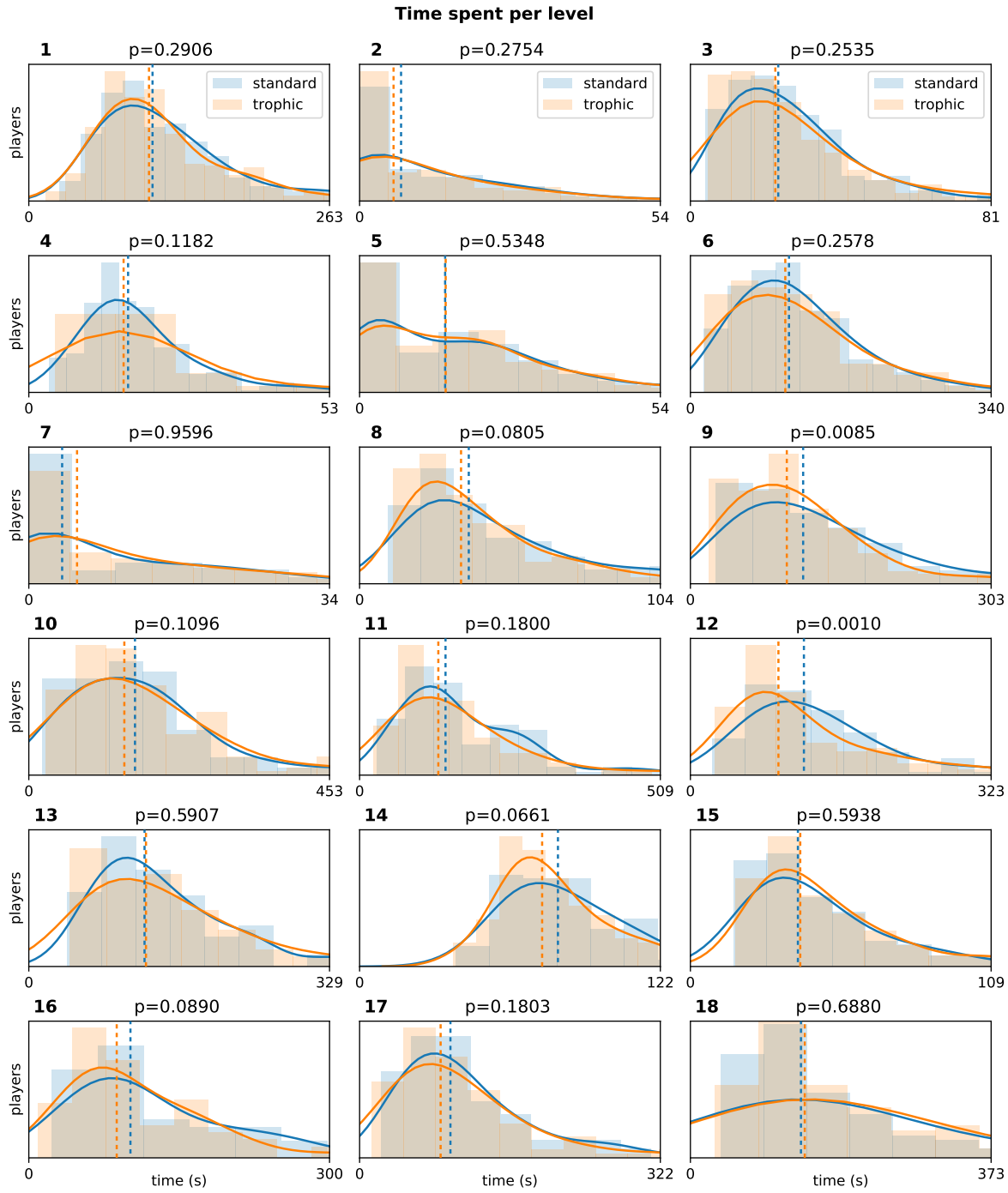


Figure 5.7: Histograms of the time taken by both sets of players for every level, with the same conventions as Figure 5.6. The index of each level is written on the top left of every plot, where the upper limit of the x-axis is set to the time at which 95% of players managed to finish.

naturally going to become especially crowded there. In these situations, fixing the y-axis causes nodes to bunch too closely together to differentiate. Standard layouts still become hairballs, but at least nodes are separated enough to not overlap too much. The remaining research world levels had little to no difference in median scores, which shows that for many tasks there is no value to be gained for adding the trophic constraint to the layout.

These results point towards a small, but clear improvement in performance for the trophic visualisation over the standard one. This is enough to recommend the method for the general application of food webs, but with the condition that care must still be taken to consider the specific use case, as it is not always going to help.

5.3.2 Objective 2

After completing Learning World, players unlock the right to compete against each other on three extra levels, as detailed in Section 5.2.2. The analysis of these levels will be more qualitative than in Learning World, but nevertheless, all three sets of results reveal interesting patterns of how players approached solving the challenges, and of which strategies were good enough to be the very best.

Each level gets its own page of plots, where Levels 1, 2, and 3 can be found in Figures 5.8, 5.9, and 5.10 respectively. The food webs of the top three players are drawn on the left of each plots, along with their names. On the right of each plot are histograms plotting some features of these top three webs in the top half, and the same features from the top 50 scores on the bottom. The top half therefore shows the best of the best strategies, and the bottom shows the general patterns of all good strategies to place relatively highly on the leaderboards.

The three levels will be examined in succession, starting with Level 1, where a very clear correlation of trophic to body mass is shown in the top 3 scores. This pattern

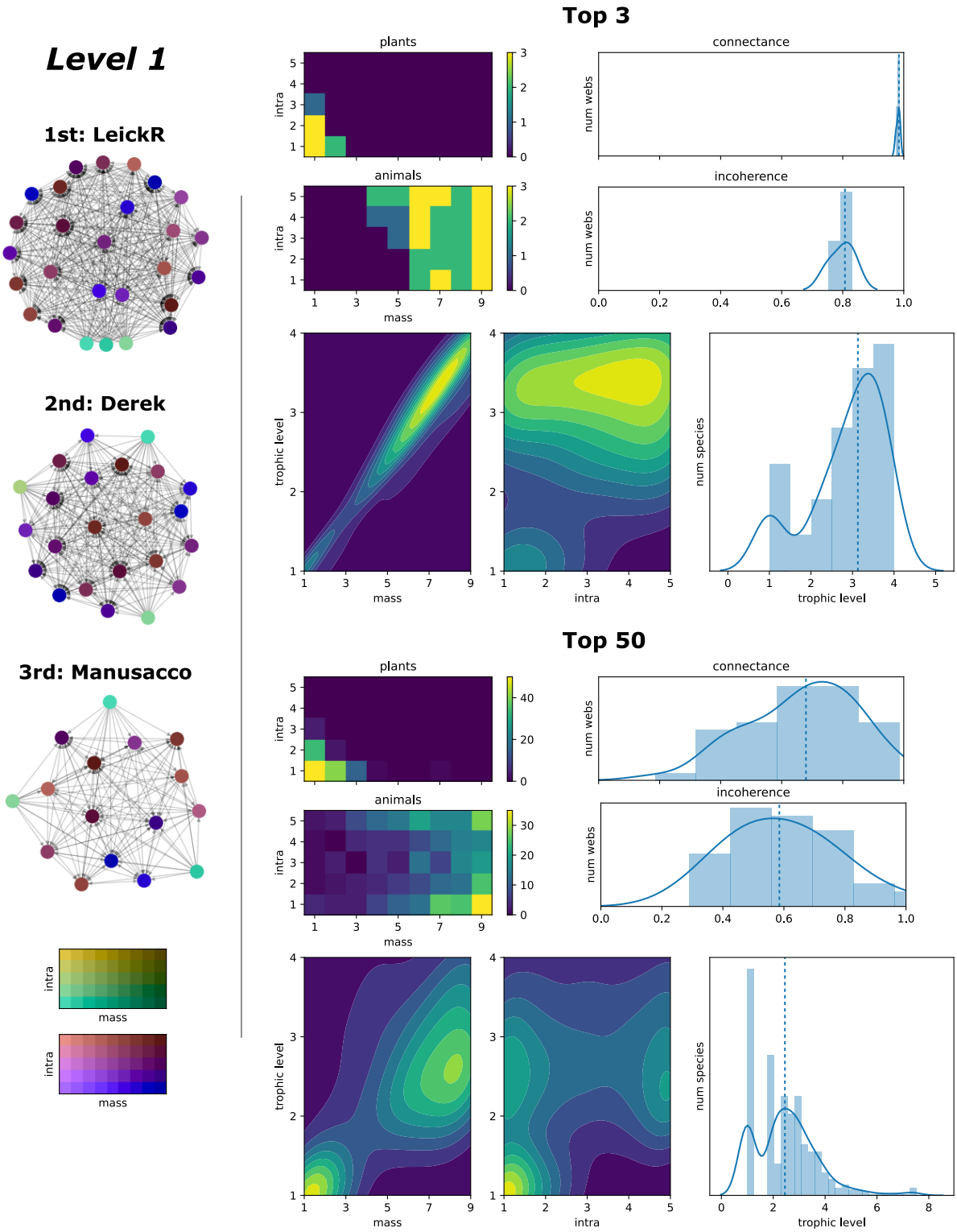


Figure 5.8: An analysis of strategies used for Level 1 of Research world. On the left are layouts of the top 3 leaderboard scores, with a colour map to show the colours of trait combinations for plants and animals. To the top-right is an analysis of these three scores with various histograms. Counter-clockwise from the top-left: the number of plants with each possible combination of traits, the same but for animals, the masses and trophic level of each species, the same for intraspecific interference, trophic levels of all species, incoherence (Equation (5.12)), and connectance. On the bottom-right are the same plots, but for the top 50 scores.

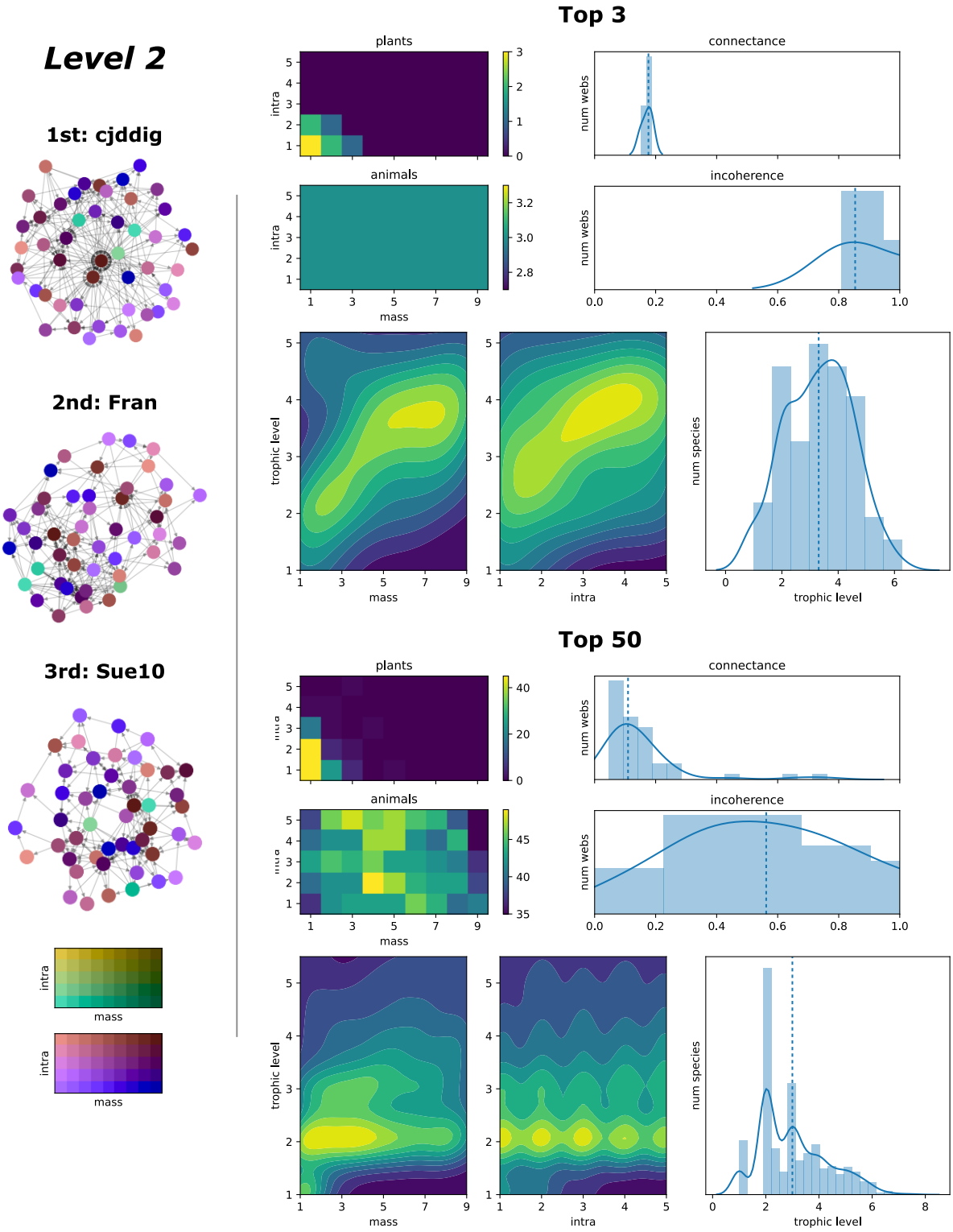


Figure 5.9: An analysis of strategies used for Level 2 of Research World, with the same conventions as Figure 5.8.

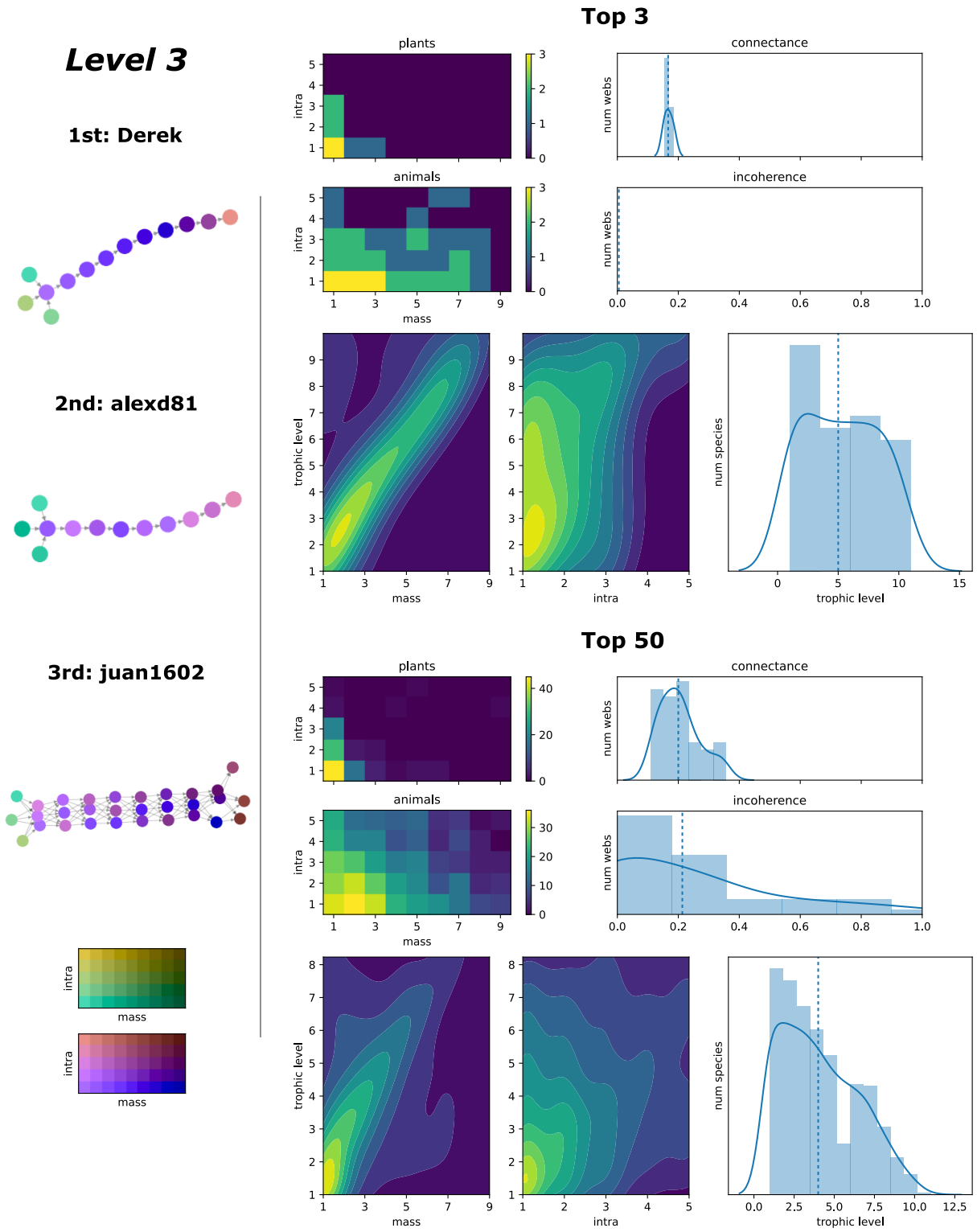


Figure 5.10: An analysis of strategies used for Level 3 of Research World, with the same conventions as Figure 5.8.

is weaker but still present for the top 50 webs. Connectance for Level 1 is very high because the score metric (Equation (5.17)) demands it, and webs are in general incoherent. From this, it can be posited that a strong correlation between trophic level and body mass is necessary for webs with high connectance to be feasible.

Level 2 removes the need for high connectance, and it is clear that players took advantage of this, as the median connectance shot all the way down to below 0.2 for the best webs, and even lower over the top 50. This is strong evidence that high connectance webs are more difficult to build, which is in line with most predictions (May, 1973; Allesina and Tang, 2012). In fact this is such a strong effect that, as shown by the completely flat histogram of animal traits, that the best players were able to make all 45 possible animals coexist. This is not only impressive, but also shows that building large and diverse ecosystems that are feasible is possible, given the right configurations. This median value for connectance is also in line with predictions for empirical food webs (Goldwasser and Roughgarden, 1997), which means that, given few constraints and minimal information, the topological structure of constructed food webs mimics that of the real world. As a caveat, the drop in connectance may also be a side-effect of high-connectance webs producing more cluttered visualisations, such that players may actively try to avoid them. It is also of note that the the best webs in Level 2 are very incoherent. This is a signal that there is a lot of omnivory in diverse ecosystems, which may suggest that omnivory plays a part in supporting large ecosystems.

Finally, the best webs for Level 3 show that for long chain lengths to exist, then coherence is likely to be necessary. The plot of mass and trophic levels shows the same pattern of heavier species at high trophic levels as before, but with more lighter species overall in the web than heavy. This is likely because lighter species are required to sustain high levels of biomass flow up to the next trophic level, and so more are needed to be a powerful 'base' for the webs. It is also interesting that

this pattern of more lighter species than heavy is what is commonly observed in nature (Fenchel, 1993), which points towards the large chain lengths of Level 3 being at least in part similar to that of natural ecosystems. There is also a clear pattern in the trait combination plots, where most species, including animals, are concentrated around low mass and low interference. This is very different from the high connectance webs in Level 1, where heavy species were preferred, and high interference is used as often as low. This is again likely because of the need for a large amount of biomass flowing up the lower levels.

It is also clear from all levels that low mass and low interference is optimal for plants. This is relatively obvious, as it is the combination that results in the highest plant abundance. There is therefore no evidence that players ran into the 'paradox of enrichment', that states that sometimes higher abundance of prey can destabilise webs (Rosenzweig, 1971).

The food webs built in Research World can also be used to examine the choice of range of interference strengths (diagonals in Equation (5.2)), which was 10^{-4} to 10^{-1}. Looking at all off-diagonal values in all interaction matrices in the 3 Research levels shows that 95% of the values lie in between the range 5×10^{-6} to 7×10^{-2}. This range is in roughly line with the results of Ho (2020), where it is shown that maximum feasibility in food webs occurs when the diagonals are around ten times larger than the off-diagonals.

5.3.3 Discussion

The work in this chapter has presented EcoBuilder: a video game for crowdsourcing mathematical ecology. It was publicly promoted from the 31st of July 2020, and the results here were taken from data collected up until the 16th of August, making up a span of one and a half months. Demographics for players of the game

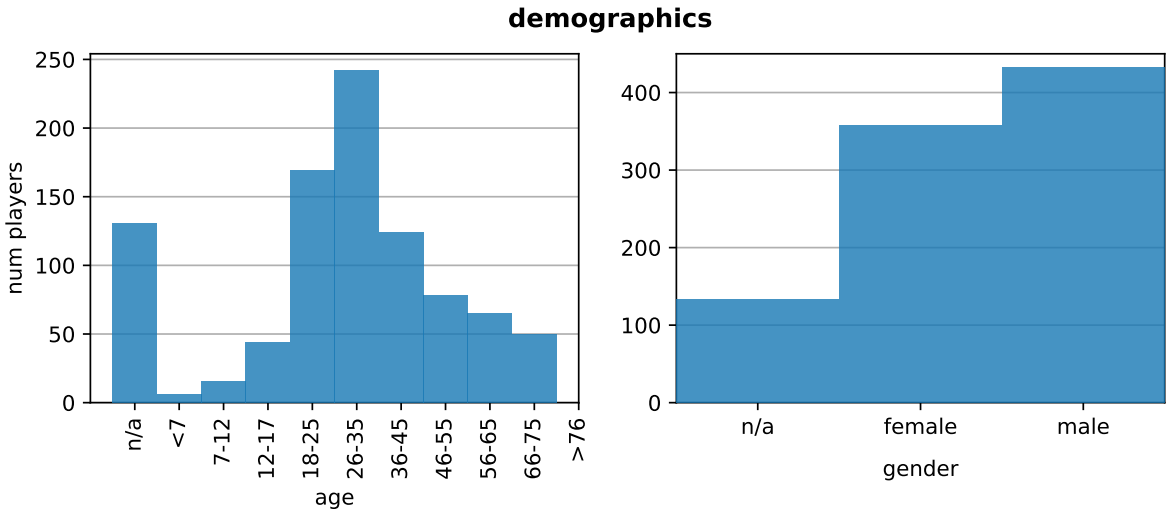


Figure 5.11: Player demographics for EcoBuilder. Most players were ages in in the 20s to 30s, but with a good mix of other age groups too. Gender was also slightly more biased towards males, with no players picking a third 'other' option.

can be seen in Figure 5.11.

There were two research outcomes planned from the game. The first was a quantitative comparison of two different visualisations, where one is a standard layout and another has y-axis positions constrained. The results of this are outlined in Section 5.3.1, where it was shown that constraining the y-axis does make a small, but significant difference to many of the tasks performed in the game. The results do not mean an exemplary recommendation for trophic constraints in all situations, but it does show small and consistent improvement on the tasks set for the player within the game.

An older version of the game also planned to make use of the 3D capabilities of the Unity game engine, by placing species in three dimensions and having players rotate the resulting node-link diagram. However, early playtests revealed that players could not digest this interface easily, and so the game was converted back to 2D.

The second outcome was to have players tackle ecological puzzles where the solutions are far from clear, in an attempt to crowdsource the intelligence of humans

for pattern recognition. It was hoped that a healthy dose of competition would motivate players to find structural patterns to build healthy ecosystems, and the results here show that this faith was well placed. Players exceeded expectations, especially on level two where many were able to make all possible species coexist. The qualitative analysis presented in Section 5.3.2, even as an exploratory and early stage study, revealed a number of patterns in player strategies, and a more precise dive into specific topologies of the top scores is sure to show even more. It is clear that the general public are a powerful resource, and that they are more than happy to contribute when presented with a challenging but accessible opportunity to do so.

As previously discussed in Section 5.1.3, the number of possible topologies for ecosystems is more than exponential, and so it is impossible for computers to perform an exhaustive search for large and healthy food webs. The high scores achieved by players were therefore something that only a crowdsourcing approach could have revealed, and the ability for players to achieve the scores they did is also strong evidence for player learning of the concepts. Future versions of the game may take advantage of this, by introducing a back-and-forth approach between players and researchers. The strategies found (such as placing heavier species at higher trophic levels) could be implemented into computer search routines to find even larger ecosystems. Another promising avenue for future versions would also be the introduction of species body temperature as another trait for the player to tweak, possibly also to model the effects of climate change on food webs.

Another important observation for future projects that involve interactive networks is that, on the small vertical screens of mobile devices, layouts can get very crowded with even a couple of dozen nodes. Any sequel that would attempt to consider even larger ecosystems, i.e. in the order of hundreds of species, would be

more suited to a desktop setting, and would likely require a more specialised visualisation than a standard force-directed layout. The idea of restricting the player to two-dimensional topologies, as defined in Section 5.1.3, could offer a neat solution to the problem, as each species would only require a position, along with two coordinates to specify its feeding range, to fully describe its incoming links. Edge bundling techniques like those studied in Chapter 4 were also deemed as too complex for the context of a mobile game, but a desktop version could open that door as well.

A priority for any sequel would also be to implement the improved layout stability methods outlined in Section 5.1.4. Even though the networks considered in the game are small, layouts that did not preserve the mental map of the player were a more common occurrence than desired, and this problem would be exacerbated even further if larger networks were involved. One potential option is to continue the crowd sourcing theme and use the method of Yuan et al. (2012), who use many human-produced layouts of subgraphs to piece together an overall pleasing layout. At the very least, more interactive exploration options would surely be of use to the player.

All source code and assets used to produce the game are open source and publicly available at `www.github.com/jxz12/EcoBuilder`.

5.4 Appendix: User interface

This section describes the user interface of EcoBuilder in detail. It is placed as an appendix to this chapter, as the content is not strictly necessary for understanding the research aims and goals presented in the main text.

There are multiple components that go into the development of any game, and



Figure 5.12: An illustrative screenshot of gameplay in EcoBuilder. The components labelled 1–8 are explained in Section 5.4.

EcoBuilder is no different. The Unity game engine (Goldstone, 2009) was chosen as the computational framework, due to its ease of compilation to various platforms, including mobile and WebGL. To illustrate the various engineering challenges behind the user interface, Figure 5.12 will be used as a reference screenshot of the gameplay performed by a player. The various moving parts are labelled in the order they will be explained. However, the best way to gain an understanding of the interface is to try it – links for downloading the game on iOS and Android can be found at `www.ecobuildergame.org`.

5.4.1 Food web visualisation

The most complex component of the gameplay is the visualisation of the ecosystem itself, as it must simultaneously display its topology and dynamical behaviour. For that reason, this whole section will be devoted to its explanation, and the following Section 5.4.2 will concern the remaining components.

As shown by label 1 in Figure 5.12, a node-link diagram is used for visualising the topology of the food web. The nodes of the diagram represent species, where their shape and colour are dependent on the two traits, size and interference (further explained in Section 5.4.2, label 6). Node colour depends on both traits simultaneously, to span a 2D plane in the 3D CIELAB colour space to maintain perceptual uniformity (Smart, Wu, and Szafir, 2019), where the planes used for plants and animals can be seen in bottom left of Figures 5.8 to 5.10. Node shapes were designed to be relatively spherical in order to resemble a classical node-link diagram, and change depending on the choice of plant or animal. The cute faces are procedurally generated using random combinations of eyes and mouths. Links are rendered with low opacity, except with moving circles that flow in the direction of biomass transfer. Such animated edges were studied by Holten, Isenberg,

et al. (2011) to be effective at showing link direction, and also recommended in Bach et al. (2017). Coloured outlines around nodes and links are also used to indicate their status, for example in Figure 5.12 a blue outline means the component cannot be edited, and a yellow outline means it is the current focus of the panel labelled by the number 6.

To show the equilibrium abundance of each species, as calculated by Equation 5.3, a small symbol is placed next to each species, which follows the same conventions as battery life indicators on mobile devices, and health bars in video games. Because this abundance can be negative (corresponding to extinction), values below zero are shown as red that only fills a small fraction of the space, and positive values as green that fills up more. An example of negative abundance can be seen in Figure 5.1. Numbers showing the exact traits of each species are also placed above the battery icons, so that the exact definition of the species can be exactly identified.

Interactive layout

The introduction of interaction adds multiple extra factors that need to be taken into consideration. The first is the question of how the player adds new edges to the graph. This is done in the simplest and most intuitive manner: the player simply drags in between the two nodes they wish to attach to each other. Since the flow of biomass is directed from prey to predator, the first intuition may be to drag from the prey to predator, however preliminary feedback showed that players found it more intuitive to think in terms of *this eats that*. The default setting in the game therefore has the player drag from predator to prey, but this is left as an option in the settings.

The Unity game engine works by providing the programmer with functions to im-

plement that will be run once per frame, but SGD is not quite fast enough to be completely computed every frame and still maintain a good frame rate. Because of this, the layout code is implemented as a thread that runs concurrently alongside the primary frame-by-frame thread, and only returns the layout to display on the screen once it is completed. SGD is fast enough to only require a couple of frames, even on mobile devices, but concurrency is still required to prevent stutter. On top of this, the local version of majorization (see Section 2.1.3) is used once per vertex per frame in order to fine tune the layout slowly into a precise minimum.

As described in Section 5.1.4, successive topologies constructed by the player are aligned using a Procrustes transformation, and will be described in detail here. The Procrustes *error* is simply defined by the sum of squared distances between two layouts

$$\text{procrustes}(\mathbf{X}, \mathbf{X}') = \sum_{i \in V} ||\mathbf{X}_i - \mathbf{X}'_i||^2. \tag{5.13}$$

The purpose of the corresponding transformation is to minimise this error, and can be performed exactly using linear methods.

Removing the translational component is performed by simply placing the barycenter of both layouts at the origin. Traditionally $\mathbf{X}'$ is then dilated, but stress is not invariant to scaling so this step is skipped here. The rotational component is then removed by differentiating the right-hand side of Equation (5.13) with respect to a rotation θ and solving for the derivative equalling zero, resulting in

$$\theta(\mathbf{x}, \mathbf{y}, \mathbf{x}', \mathbf{y}') = \tan^{-1}\left(\frac{\sum_i (\mathbf{y}_i \mathbf{x}'_i - \mathbf{x}_i \mathbf{y}'_i)}{\sum_i (\mathbf{x}_i \mathbf{x}'_i - \mathbf{y}_i \mathbf{y}'_i)}\right) \tag{5.14}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the two dimensions of the layout $\mathbf{X}$, and similarly for $\mathbf{X}'$. Since the dilation step was skipped, this process is performed for both the new layout and its reflection, where the result with a lower value of Equation (5.13) is chosen.

Note that usually in the literature, the square root of this value is taken as the final statistic. If the sets of vertices differ between topologies, then their intersection is instead compared.

To transition smoothly between aligned layouts, the `SmoothDamp()` function provided by Unity is used, which attaches a critically damped spring between an object and its target position, to avoid jerky movement by maintaining continuity (Kirmse, 2004). Connected components are layed out separately, and simply placed horizontally adjacent to each other such to avoid overlap.

Constraining the y-axis

A domain-specific feature of food webs is that any valid topology allows for the trophic level to be calculated for each species, following the process outlined in Section 5.1.3. For this reason, an additional constraint to the layout is included, that constrains the y-axis to the trophic level of the species.

The diagonal dominance of the Laplacian matrix to be inverted in order to solve for trophic levels allows for the Jacobi method to be used. Expressing the trophic level Equation (5.11) as

$$\mathbf{y} = \mathbf{L}^{-1}\mathbf{b}, \tag{5.15}$$

ordinary Jacobi solves for $\mathbf{y}$ by assigning

$$\mathbf{y}_i' = \frac{1}{\mathbf{L}_{ii}}\left(\mathbf{b}_i - \sum_{j \neq i} \mathbf{L}_{ij}\mathbf{y}_i\right) \tag{5.16}$$

for each vertex i (Young, 2014). In this case, $\mathbf{b}$ is simply a column vector of ones. An iterative method was chosen because the topology of the graph will only change by a single edge at a time, and so previously calculated trophic levels is a good approximation for subsequent values. From experience, the number of iterations

required for convergence is almost always fewer than ten. Since the matrix $\mathbf{L}$ contains an entry for every edge, the complexity per iteration is $\mathcal{O}(|E|)$, as opposed to solving the system exactly. Equation (5.15) does not have a solution if any component has no vertex with an in-degree of zero (i.e. no primary producers). In this case, trophic level computations are simply ignored.

A problem here is that stress is more difficult to optimise, because there is now only one degree of freedom for vertices to move around in. The solution found was to increase the maximum value of μ in Equation (2.21) to 1.1, as well as adding five extra initialisation iterations at the beginning of the optimisation. This is however an ad hoc solution and future work should involve testing variations of this systematically. The aforementioned Procrustes statistic also skips the rotation step in this case, as any rotation would break the orientation of these constraints. An example of the same food web visualised with and without this constraint is shown back in Figure 5.5, where it can be seen that the constrained version tends to point edges upwards.

An important question is whether constraining the y-axis is helpful to the user in the first place. Intuition says the constraint will largely make edges point upwards, which should help to preserve the player's mental map of the flow of biomass throughout the system. The opposing argument is that layouts will have higher stress, and so may become more crowded. This question will be systematically tested by the game. Half of the players will be given an standard layout and the other half a constrained layout, and data collected from the game will be used to study which is more effective. More detail on this experiment is outlined in Section 5.2.

Focus

A final addition to Research world is another tool for navigating the topology of food webs. If a species is tapped twice, the layout transitions into a mode where only the direct connections to that species are visible, where its prey are positioned below it and its predators above. The topology can then be navigated in this mode, by hopping from one node to another by again tapping. This extra feature was necessary because as the number of species goes up, layouts can become crowded and difficult such that it is difficult to see fine details. The ability to focus attention on a single node helps to alleviate this.

5.4.2 Heads-up display

The remaining labels on Figure 5.12 point to elements of a 2D overlay, upon which exists what is known as a heads-up display (HUD). The idea of a HUD is now ubiquitous to video game interfaces, but its name originated from military aviation, where the pilot would need to have extra information available whilst leaving their view unobstructed. Its purpose in the context of games is the same, where it gives the player the remaining information required to complete the game, whilst leaving the main visualisation as visible as possible.

Label 2: Constraints

Different levels offer different scenarios, with specific topological constraints shown on this first panel. The top two icons show the maximum numbers of plants and animals the player may introduce, and the bottom two show the number of edges and the maximum chain length. Chain length is defined for each plant as

zero, and each animal as the shortest path from any plant to it. These are all used throughout the game as conditions required to pass each level.

Earlier versions of the game also included maximum loop length as an extra constraint, but there does not exist a polynomial time algorithm to search for all loops in a graph. Johnson's algorithm (Johnson, 1975) was attempted for this, but its running time grew to unacceptable levels, and so it was cut from the final release of the game.

Label 3: Score

This panel displays the player's score, as well as how many stars they have obtained. To obtain the first star, the only conditions are that all species coexist and that there is a single connected component. The second and third stars require the player to reach certain numerical score thresholds as a further challenge for skilled players. The current score is shown as the number in the pale brown subpanel, and the target is shown as the green number below.

Many different metrics were tried and tested for use as a score, most of which were unsuitable. For example, the total abundance every species was a good candidate, but this is maximised by having only plants in the food web, as any animals simply lose too much biomass via the efficiency factor e_0 in Equation (5.8). The score should instead continuously increase as the food web becomes more complex. Another related option was to take the total *flux* of the food web, defined as the total amount of biomass flow through every interaction. This is defined here as the sum of the positive off-diagonals in Equation (5.5).

One definition of quantitative complexity is $\sigma\sqrt{nC}$, where σ is the standard deviation of the off-diagonals of the community matrix in Equation 5.5, n is the number of species, and C is the connectance, defined as the proportion of realised to pos-

sible interactions such that $0 \leq C \leq 1$. Specifically this is $C = |E|/(n(n-1)/2)$. This quantity was shown by May to negatively correlate with the probability of stability as defined in Equation (5.6), and acted as the springboard for the aforementioned complexity vs. stability debate. This criterion has been taken and updated to more accurately match real ecosystem structure (Allesina and Tang, 2012; Tang, Pawar, and Allesina, 2014), but all contain the components σ, n, and C. Unfortunately none of these metrics were able to fit the purposes of a game, because the factor σ is not very interpretable, and initial feedback showed that players found it difficult to work out why it would go up or down. Leaving out σ and only including n and C does not work either, since they are both structural measures and so do not capture any dynamics of the system.

The final formulation settled upon was a compromise between the aforementioned options: the factor σ is replaced by the total abundance and the square root removed, to construct a final score of

$$\text{score} = \left(\sum_i^n \mathbf{z}_i^* \right) nC \tag{5.17}$$

where $\mathbf{z}^*$ is from Equation (5.3). This alternative was found to be interpretable and goes up as the food web grows in size.

Label 4: Level

The square in the top right of the HUD is a button that represents the level currently being played. When the player has earned the first star of a level, this button can be pressed to pass the level and unlock the next. The appearance of this button before the level has been passed can be seen in Figure 5.1, where pressing it gives the option to quit the game or reset the level. The screen shown to the player upon completing a level is shown in Figure 5.13, left.

Label 5: Help

This panel is present throughout the game, in both menu screens and during a level playthrough. It was necessary to not overfill this box for two reasons: the first is that the target platform for the game is mobile devices, so space is limited; the second is that in preliminary playtests, players tended to skip any long blocks of text. Some players would even skip all text, regardless of length, and so care had to be taken to use a 'learn-by-doing' approach as often as possible. For that reason, multiple levels that act as heavily guided and constrained tutorials were included.

The green arrow on the edge of the panel can be used to hide or show the text at will. If the player does not find the solution to a level after two minutes, the text also shows itself to reveal a hint. Further playtests showed that the sometimes counter-intuitive behaviour of the simulation can be frustrating to players, so giving extra help after a delay was a necessary step, a technique often used by game developers to ensure that players get stuck as rarely as possible.

Label 6: Inspector

This panel is known as the inspector, and is the avenue through which the player views and changes the traits of their species. These two traits, body mass and interference, take a wide range of values: mass goes from one gram to one thousand kilograms, therefore spanning six orders of magnitude; interference goes from 10^{-4} to 10^{-1}, spanning three orders of magnitude. This range for mass was chosen to capture species of almost all sizes in the natural world. Smaller species do exist and are vital to Earth's ecosystem (such as bacteria and the smallest insects) but they are left as out of scope for the game. The range of interference values was set to span this range, as preliminary playtests showed that species with any larger

interference tended to be very difficult to keep alive at all, and species with any smaller interference tended to destabilise the rest of the food web. Preliminary tests also showed that allowing the player to choose interference on top of body mass was too much information for players to begin with. The ability to tweak interference by hand is therefore not unlocked until the second set of levels known as Research World is reached (see Section 5.2). Further discussion of interference values can be found in Section 5.3.2.

Both traits are also set according to a log scale, so that all orders of magnitude are accessible. The sliders themselves are constrained to only take discrete values, so that the player knows the exact values being used. This also means that values can be displayed using a single number next to the species, as shown in the screenshot in Figure 5.12.

The button to the upper-left of the panel can be used to remove unwanted species from the food web. It is placed next to the name of the species, which is randomly generated by joining together sequences of syllables.[4] The player may also input their own name for each species, if they wish to do so.

Label 7: Initiator

This panel contains two buttons that are used to initiate the addition of a new species to their ecosystem. It is at this point at which the binary choice between plant and animal is chosen and fixed, which not only makes sense from a biological standpoint, but is also necessary in terms of topology, because plants must have an in-degree of zero i.e. they cannot be predators. This is not strictly true for the real world, because species such as the Venus flytrap are in fact carnivorous plants. There are even animals that can perform photosyntesis, such as the sea

[4] Credit for the random name generation algorithm goes to `www.fantasynamegenerators.com`, accessed 13/8/20.

slug, which can 'steal' the ability to produce chloroplasts from the algae it eats (Rumpho et al., 2008). These such creatures, as remarkable as they are, were considered rare enough to leave out of scope for EcoBuilder.

At this point it is worth mentioning that the number of plants was also limited to a maximum of three in any given level, and usually fewer. This was necessary here because there is no explicit competition between plants, i.e. a $(-,-)$ pair in the interaction matrix, and so there is no penalty at all for adding more of them. This decision was made because of the difficulty behind parameterising these extra interactions, and so the limiting of the number of plants was instead used. Other plant growth models such as the *chemostat* model (Sommer, 1983) were also considered, but not utilised in the final version.

Label 8: Recorder

The final component of the gameplay in EcoBuilder is that ability to undo and redo actions, a feature now considered standard among most consumer applications. The arrow pointing left goes backward to undo an action, and the arrow pointing right goes forward to redo a previously undone action, as is conventional. It is implemented simply using a stack, and so undone actions are lost forever if a new action is performed. The criteria for being an action is a topological change, i.e. the addition or removal of vertices or edges, or a change in value for the trait of a species.

The entire sequence of actions performed by the player is recorded, and when the level is completed it is transformed into a string to sent to a server, along with other playthrough data. The remaining contents of this data will be explained in the following section.

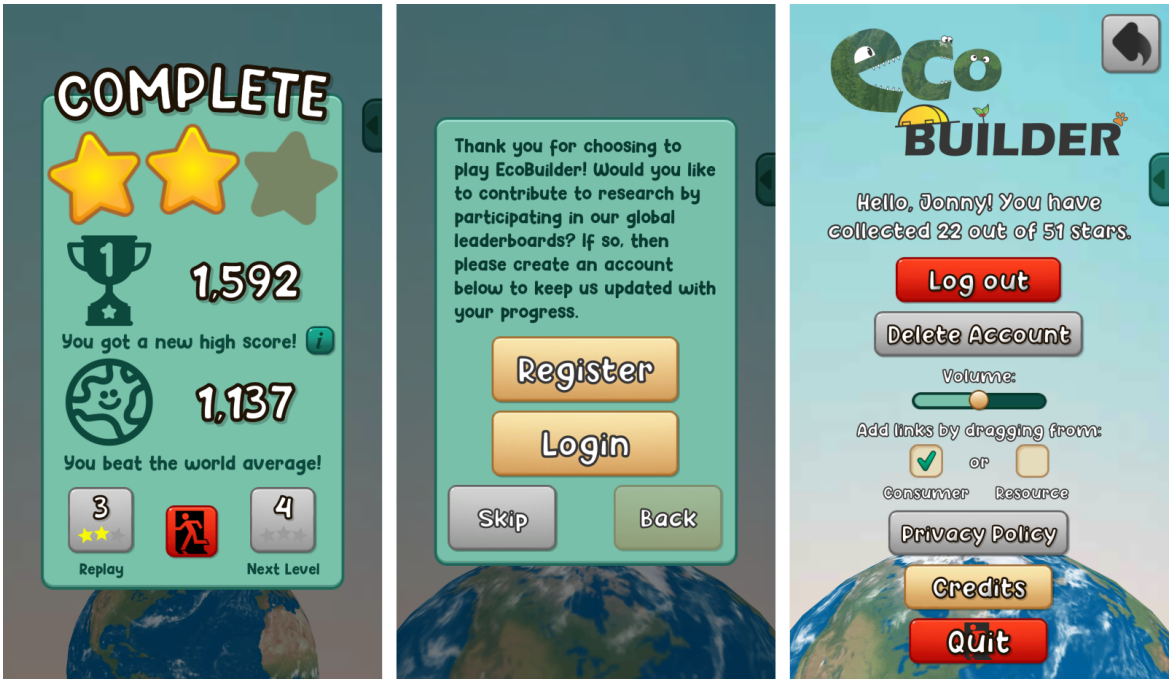


Figure 5.13: Screenshots to illustrate player registration and data collection. From left to right: the report screen shown when a level is completed, the registration page for player accounts, and the settings page for account details.

5.4.3 Data collection

Figure 5.13, left, depicts the report card shown to the player when they complete a level. It tells the player the number of stars they achieved, their score, and the median score from all high scores throughout the world. The median score is shown in order to motivate the player to strive for higher scores, as a small competitive incentive. The small button to the middle-right of the panel can also be pressed to show the exact decomposition of their score, as defined by Equation 5.17, but scaled to fall within a reasonable range of values that make sense as a score. The player can then replay the level or continue to the next.

When this report card panel is shown, a POST request is sent to the server containing multiple fields, including: a timestamp, the score, each component of the parameterisation defined by Equation 5.2, a list of timed actions performed by the player, and a list of device metadata such as operating system and resolution.

These fields are received by the server and stored in an SQL database for future analysis.

The middle screenshot in Figure 5.13 shows the registration page that is shown the first time the application is opened. An account is required for two purposes: the first is to ensure that the data being collected can be matched to a single player, even if they decide to switch devices; the second is to have them tick a box to indicate that they are aware of the general data protection regulation (GDPR) their data will be held under. Note that since there is no 'sensitive' personal data being collected, such as ethnicity or political opinions, a tick box is not strictly necessary. Some general demographic information, such as age and education background, is optionally collected at registration. The player may also delete their account whenever they wish, as shown by the settings page to the right of Figure 5.13. Here they may also change game settings such as volume or drag direction. They may also view the GDPR privacy policy in full, and a credits page to thank the lab members who kindly gave their time to help playtest the game.

Conclusion

This thesis has explored the topic of network visualisation using the 'join-the-dots' representation known as the node-link diagram. Three chapters studied three distinct subjects within the context of this representation. Chapter 2 considered the problem of node layout, where the idea of force-directed algorithms was framed in the general context of optimisation, which brings a variety of algorithms into a shared perspective. The scope was then focused on the popular energy function known as stress, where an algorithm known as stochastic gradient descent was applied to optimise stress more efficiently than the previous state-of-the-art algorithm, as shown through an experimental study. It was also applied to an approximation of the stress summation that allows the method to scale to large graphs. Constrained layout and general multidimensional scaling were also briefly explored as further capabilities.

Chapter 3 dealt with the concept of allowing links to be curved, in order to untangle the hairballs that often result from using force-directed algorithms on real-world data. The method known as hierarchical edge bundling was the main focus, which classically uses hierarchical metadata to inform bundling; the work here studied the situation where this metadata must be inferred. Many options for this inference were compared through an experimental study on datasets with a ground truth included, and recommendations were made for application in practice. Chapter 4 compared this method to a topologically lossless version of edge

bundling, known as power-confluent drawing. The hierarchical nature of the lossless method is leveraged to solve some technical issues with a past implementation, and some small improvements are presented to improve the greedy agglomeration preprocessing step.

The thesis is rounded out by Chapter 5, which presents the design and development of EcoBuilder, a research-oriented video game that allows the player to build their own simulated ecosystem. The node-link diagram representation was used to visualise both the structural topology and dynamical behaviour of the constructed food webs. This choice of representation was studied by performing a controlled test between two randomly allocated groups of players, where one group was presented a layout with the y-axis of nodes constrained to the trophic level of its corresponding species. The effect of this extra constraint was shown to be statistically significant on player performance, but not in all scenarios and so caution was recommended. A secondary research outcome was to crowdsource good strategies from players to build diverse and healthy ecosystems, by providing a global leaderboards for players to compete with each other on. An exploratory analysis of the configurations assembled by top players revealed strong patterns of species bodysizes correlating with trophic level, with various other features also shown to be of interest.

Visualisation is a rich topic filled with interesting problems, and network visualisation is no exception. From algorithmic challenges all the way up to questions of human psychology, there exists a wide variety of research opportunities to explore at all levels of abstraction. The overarching purpose of visualisation is also to provide tools for practitioners to analyse real-world data, and so the leap from theory to practice is tangible, and the impact of work done immediate. A perfect example of this is EcoBuilder itself, where the applied domain is food webs. Without research into the underlying algorithms that drive visualisation, such a use

case would not be possible. Without such use cases, the dynamics of the systems within these domains would remain opaque.

If nothing else, visualisation can also be beautiful. It is the hope that the work in this thesis goes at least a small way towards making this beauty visible for more people than before.

Bibliography

Agutter, P. S. and D. N. Wheatley (2004). "Metabolic scaling: Consensus or controversy?" In: *Theoretical Biology and Medical Modelling* 1, pp. 1–11.

Ahmed, R. et al. (2020). "Graph drawing via gradient descent, $(GD)^2$". In: *International Symposium on Graph Drawing*, pp. 1–17.

Ahnert, S. E. (2014). "Generalised power graph compression reveals dominant relationship patterns in complex networks". In: *Scientific Reports* 4, p. 4385.

Albano, J. A. and D. W. Messinger (2012). "Euclidean commute time distance embedding and its application to spectral anomaly detection". In: *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery*. Vol. 8390, 83902G.

Allesina, S. and S. Tang (2012). "Stability criteria for complex ecosystems". In: *Nature* 483.7388, pp. 205–208.

Archambault, D. (2009). "Structural differences between two graphs through hierarchies". In: *Graphics Interface*, pp. 87–94.

Archambault, D., T. Munzner, and D. Auber (2008). "GrouseFlocks: Steerable exploration of graph hierarchy space". In: *IEEE Transactions on Visualization and Computer Graphics* 14.4, pp. 900–913.

Archambault, D. and H. C. Purchase (2012). "Mental map preservation helps user orientation in dynamic graphs". In: *International Symposium on Graph Drawing*. Springer, pp. 475–486.

– (2013). "The "map" in the mental map: Experimental results in dynamic graph drawing". In: *International Journal of Human-Computer Studies* 71.11, pp. 1044–1055.

Archambault, D. and H. C. Purchase (2016). "Can animation support the visualisation of dynamic graphs?" In: *Information Sciences* 330, pp. 495–509.

Archambault, D., H. C. Purchase, and B. Pinaud (2011). "Animation, small multiples, and the effect of mental map preservation in dynamic graphs". In: *IEEE Transactions on Visualization and Computer Graphics* 17.4, pp. 539–552.

Argyriou, E. N., M. A. Bekos, and A. Symvonis (2010). "Maximizing the total resolution of graphs". In: *International Symposium on Graph Drawing*. Springer, pp. 62–67.

Arjovsky, M., S. Chintala, and L. Bottou (2017). "Wasserstein GAN". In: *arXiv preprint*. arXiv: 1701.07875.

Armstrong, R. A. and R. McGehee (1980). "Competitive exclusion". In: *The American Naturalist* 115.2, pp. 151–170.

Bach, B. et al. (2017). "Towards unambiguous edge bundling: Investigating confluent drawings for network visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1, pp. 541–550.

Bae, J. A. et al. (2018). "Digital museum of retinal ganglion cells with dense anatomy and physiology". In: *Cell* 173.5, pp. 1293–1306.

Bar-Joseph, Z., D. K. Gifford, and T. S. Jaakkola (2001). "Fast optimal leaf ordering for hierarchical clustering". In: *Bioinformatics* 17.suppl_1, S22–S29.

Barabás, G., M. J. Michalska-Smith, and S. Allesina (2017). "Self-regulation and the stability of large ecological networks". In: *Nature Ecology & Evolutionion* 1.12, pp. 1870–1875.

Barnes, J. and P. Hut (1986). "A hierarchical O(N log N) force calculation algorithm". In: *Nature* 324.6096, pp. 446–449.

Bartel, G. et al. (2010). "An experimental evaluation of multilevel layout methods". In: *International Symposium on Graph Drawing*. Springer, pp. 80–91.

Battaglia, P. W. et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint*, pp. 1–40. arXiv: 1806.01261.

Behrisch, M. et al. (2016). "Matrix reordering methods for table and network visualization". In: *Computer Graphics Forum*. Vol. 35. 3. Wiley Online Library, pp. 693–716.

Blondel, V. D. et al. (2008). "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10, P10008.

Börsig, K., U. Brandes, and B. Pasztor (2020). "Stochastic gradient descent works really well for stress minimization". In: *International Symposium on Graph Drawing*, pp. 1–8.

Bostock, M., V. Ogievetsky, and J. Heer (2011). "D^3 data-driven documents". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12, pp. 2301–2309.

Bottou, L. (1998). "Online learning and stochastic approximations". In: *On-line Learning in Neural Networks* 17.9, p. 142.

– (2012). "Stochastic gradient descent tricks". In: *Neural Networks: Tricks of the Trade*. Springer, pp. 421–436.

Brandes, U. (2001a). "A faster algorithm for betweenness centrality". In: *Journal of Mathematical Sociology* 25.2, pp. 163–177.

– (2001b). "Drawing on physical analogies". In: *Drawing Graphs: Methods and Models*, pp. 71–86.

Brandes, U., D. Delling, et al. (2007). "On modularity clustering". In: *IEEE Transactions on Knowledge and Data Engineering* 20.2, pp. 172–188.

Brandes, U. and M. Mader (2011). "A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing". In: *International Symposium on Graph Drawing*. Springer, pp. 99–110.

Brandes, U. and C. Pich (2007). "Eigensolver methods for progressive multidimensional scaling of large data". In: *International Symposium on Graph Drawing*. Berlin, Heidelberg: Springer, pp. 42–53.

– (2008). "An experimental study on distance-based graph drawing". In: *International Symposium on Graph Drawing*. Springer, pp. 218–229.

– (2011). "More flexible radial layout". In: *Journal of Graph Algorithms and Applications* 15.1, pp. 157–173.

Brandes, U. and D. Wagner (1997). "A Bayesian paradigm for dynamic graph layout". In: *International Symposium on Graph Drawing*. Springer, pp. 236–247.

Breiger, R. L. and P. E. Pattison (1986). "Cumulated social roles: The duality of persons and their algebras". In: *Social Networks* 8.3, pp. 215–256.

Brose, U., R. J. Williams, and N. D. Martinez (2006). "Allometric scaling enhances stability in complex food webs". In: *Ecology Letters* 9.11, pp. 1228–1236.

Brown, J. H. et al. (2004). "Toward a metabolic theory of ecology". In: *Ecology* 85.7, pp. 1771–1789.

Cattin, M.-F. et al. (2004). "Phylogenetic constraints and adaptation explain food-web structure". In: *Nature* 427.6977, pp. 835–839.

Cauchy, A. (1847). "Méthode générale pour la résolution des systemes d'équations simultanées". In: *Comp. Rend. Sci. Paris* 25.1847, pp. 536–538.

Christensen, V. and C. J. Walters (2004). "Ecopath with Ecosim: Methods, capabilities and limitations". In: *Ecological modelling* 172.2-4, pp. 109–139.

Chrobak, M. and T. H. Payne (1995). "A linear-time algorithm for drawing a planar graph on a grid". In: *Information Processing Letters* 54.4, pp. 241–246.

Clarkson, K. L. (2006). "Nearest-neighbor searching and metric space dimensions". In: *Nearest-neighbor Methods for Learning and Vision: Theory and Practice*, pp. 15–59.

Cohen, J. E., F. Briand, and C. M. Newman (2012). *Community food webs: Data and theory*. Vol. 20. Springer.

Colyvan, M. and L. R. Ginzburg (2003). "The Galilean turn in population ecology". In: *Biology and Philosophy* 18.3, pp. 401–414.

Cooper, S. et al. (2010). "Predicting protein structures with a multiplayer online game". In: *Nature* 466.7307, pp. 756–760.

Cormen, T. H. et al. (2009). "Graph algorithms". In: *Introduction to algorithms*. MIT press, pp. 587–768.

Cox, T. F. and M. A. A. Cox (2000). *Multidimensional scaling*. 2nd ed. CRC Press.

Cui, H., J. Shen, et al. (2020). "Night sky brightness monitoring network in Wuxi, China". In: *Journal of Quantitative Spectroscopy and Radiative Transfer*, pp. 1–11.

Cui, W., H. Zhou, et al. (2008). "Geometry-based edge clustering for graph visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6, pp. 1277–1284.

Darken, C., J. Chang, and J. Moody (1992). "Learning rate schedules for faster stochastic gradient search". In: *Neural Networks for Signal Processing*. Vol. 2, pp. 1–11.

Davidson, R. and D. Harel (1996). "Drawing graphs nicely using simulated annealing". In: *ACM Transactions on Graphics* 15.4, pp. 301–331.

Davis, A., B. B. Gardner, and M. R. Gardner (2009). *Deep South: A social anthropological study of caste and class*. Univ of South Carolina Press.

Davis, T. A. and Y. Hu (2011). "The University of Florida sparse matrix collection". In: *ACM Transactions on Mathematical Software* 38.1, pp. 1–25.

De Coulomb, C.-A. (1785). "Premier memoire sur l'électricité et le magnétisme". In: *Histoire de l'Académie Royale des Sciences*. Paris, pp. 569–577.

De Fraysseix, H., J. Pach, and R. Pollack (1990). "How to draw a planar graph on a grid". In: *Combinatorica* 10.1, pp. 41–51.

De Leeuw, J. (1988). "Convergence of the majorization method for multidimensional scaling". In: *Journal of Classification* 5.2, pp. 163–180.

De Silva, V. and J. B. Tenenbaum (2004). *Sparse multidimensional scaling using landmark points*. Tech. rep. Stanford University, pp. 1–41.

Defays, D. (1977). "An efficient algorithm for a complete link method". In: *The Computer Journal* 20.4, pp. 364–366.

Dell, A. I., S. Pawar, and V. M. Savage (2014). "Temperature dependence of trophic interactions are driven by asymmetry of species responses and foraging strategy". In: *Journal of Animal Ecology* 83.1, pp. 70–84.

DeLong, J. P. et al. (2010). "Shifts in metabolic scaling, production, and efficiency across major evolutionary transitions of life". In: *Proceedings of the National Academy of Sciences* 107.29, pp. 12941–12945.

Dickerson, M. et al. (2005). "Confluent drawings: Visualizing non-planar diagrams in a planar way". In: *Journal of Graph Algorithms and Applications* 9.1, p. 31.

Dobson, A. P. (2014). "Yellowstone wolves and the forces that structure natural systems". In: *PLoS Biology* 12, e1002025.

Dougoud, M. et al. (2018). "The feasibility of equilibria in large ecosystems: A primary but neglected concept in the complexity-stability debate". In: *PLoS Computational Biology* 14.2, pp. 1–18.

Dwyer, T. (2009). "Scalable, versatile and simple constrained graph layout". In: *Computer Graphics Forum* 28.3, pp. 991–998.

Dwyer, T. et al. (2014). "Improved optimal and approximate power graph compression for clearer visualisation of dense graphs". In: *IEEE Pacific Visualization Symposium*. IEEE, pp. 105–112.

Eades, P., W. Lai, et al. (1991). "Preserving the mental map of a diagram". In: *International Conferences on Computational Graphics and Visualization Techniques*, pp. 34–43.

Eades, P. (1984). "A heuristic for graph drawing". In: *Congressus numerantium* 42, pp. 149–160.

Eades, P. and S.-h. Hong (2012). "How to draw a graph, revisited". In: *Expanding the Frontiers of Visual Analytics and Visualization*, pp. 111–126.

Egenfeldt-Nielsen, S., J. H. Smith, and S. P. Tosca (2019). *Understanding video games: The essential introduction*. Routledge.

Eklöf, A. et al. (2013). "The dimensionality of ecological networks". In: *Ecology Letters* 16.5. Ed. by J. Dunne, pp. 577–583.

Emmerson, M. and J. M. Yearsley (2004). "Weak interactions, omnivory and emergent food-web properties". In: *Proceedings of the Royal Society of London* 271.1537, pp. 397–405.

Enright, A. J., S. van Dongen, and C. A. Ouzounis (2002). "An efficient algorithm for large-scale detection of protein families". In: *Nucleic Acids Research* 30.7, pp. 1575–1584.

Eppstein, D., M. T. Goodrich, and J. Y. Meng (2007). "Confluent layered drawings". In: *Algorithmica* 47.4, pp. 439–452.

– (2005). "Delta-confluent drawings". In: *International Symposium on Graph Drawing*. Springer, pp. 165–176.

Eppstein, D., D. Holten, et al. (2013). "Strict confluent drawing". In: *International Symposium on Graph Drawing*. 1. Springer, pp. 352–363.

Evans, T. S. and R. Lambiotte (2009). "Line graphs, link partitions, and overlapping communities". In: *Physical Review E* 80.1, p. 016105.

Evans, T. S. (2010). "Clique graphs and overlapping communities". In: *Journal of Statistical Mechanics: Theory and Experiment* 2010, P12037.

Fáry, I. (1948). "On straight line representation of planar graphs". In: *Acta Sci. Math. (Szeged)* 11, pp. 229–233.

Fenchel, T. (1993). "There are more small than large species?" In: *Oikos*, pp. 375–378.

Fortunato, S. and D. Hric (2016). "Community detection in networks: A user guide". In: *Physics Reports* 659, pp. 1–44.

Frick, A., A. Ludwig, and H. Mehldau (1995). "A fast adaptive layout algorithm for undirected graphs". In: *International Symposium on Graph Drawing*, pp. 388–403.

Friedman, J., T. Hastie, and R. Tibshirani (2001a). "Local methods in high dimensions". In: *The elements of statistical learning*. Springer series in statistics New York, pp. 22–27.

– (2001b). "Prototype methods and nearest-neighbors". In: *The elements of statistical learning*. Springer series in statistics New York, pp. 459–484.

Fruchterman, T. M. J. and E. M. Reingold (1991). "Graph drawing by force-directed placement". In: *Software: Practice and Experience* 21.11, pp. 1129–1164.

Fulcher, L. P. and B. F. Davis (1976). "Theoretical and experimental study of the motion of the simple pendulum". In: *American Journal of Physics* 44.1, pp. 51–55.

Gansner, E. R., Y. Hu, and S. North (2013). "A maxent-stress model for graph layout". In: *IEEE Transactions on Visualization and Computer Graphics* 19.6, pp. 927–940.

Gansner, E. R., Y. Hu, S. North, and C. Scheidegger (2011). "Multilevel agglomerative edge bundling for visualizing large graphs". In: *IEEE Pacific Visualization Symposium*, pp. 187–194.

Gansner, E. R., Y. Koren, and S. North (2004). "Graph drawing by stress majorization". In: *International Symposium on Graph Drawing*. Springer, pp. 239–250.

Ge, R. et al. (2015). "Escaping from saddle points—online stochastic gradient for tensor decomposition". In: *Conference on Learning Theory*, pp. 797–842.

Ghani, S., N. Elmqvist, and J. S. Yi (2012). "Perception of animated node link diagrams for dynamic graphs". In: *Computer Graphics Forum*. Vol. 31. 3pt3. Wiley Online Library, pp. 1205–1214.

Ghoniem, M., J.-D. Fekete, and P. Castagliola (2004). "A comparison of the readability of graphs using node-link and matrix-based representations". In: *IEEE Symposium on Information Visualization*, pp. 17–24.

Girvan, M. and M. E. J. Newman (2002). "Community structure in social and biological networks". In: *Proceedings of the National Academy of Sciences* 99.12, pp. 7821–7826.

Goldstone, W. (2009). *Unity game development essentials*. Packt Publishing Ltd.

Goldwasser, L. and J. Roughgarden (1997). "Sampling effects and the estimation of food web properties". In: *Ecology* 78.1, pp. 41–54.

Good, P. I. (2006). "Testing Hypotheses". In: *Permutation, parametric, and bootstrap tests of hypotheses*. Springer Science & Business Media, pp. 33–63.

Goodfellow, I., Y. Bengio, and A. Courville (2016). "Optimization for training deep models". In: *Deep Learning*. MIT Press, pp. 271–325.

Goodfellow, I., J. Pouget-Abadie, et al. (2014). "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Gürbüzbalaban, M., A. Ozdaglar, and P. A. Parrilo (2019). "Why random reshuffling beats stochastic gradient descent". In: *Mathematical Programming*, pp. 1–36.

Gurnell, J. et al. (2004). "Alien species and interspecific competition: Effects of introduced eastern grey squirrels on red squirrel population dynamics". In: *Journal of Animal Ecology* 73.1, pp. 26–35.

Guth, A. H. (2003). "Time since the beginning". In: *arXiv preprint*, pp. 1–12. arXiv: 0301199 [astro-ph].

Habib, M., F. de Montgolfier, and C. Paul (2004). "A simple linear-time modular decomposition algorithm for graphs, using order extension". In: *Scandinavian Workshop on Algorithm Theory*. Springer, pp. 187–198.

Hachul, S. and M. Junger (2004). "Drawing large graphs with a potential-field-based multilevel algorithm". In: *International Symposium on Graph Drawing*. Springer, pp. 285–295.

Halko, N., P.-G. Martinsson, and J. A. Tropp (2011). "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Review* 53.2, pp. 217–288.

Hall, K. M. (1970). "An r-dimensional quadratic placement algorithm". In: *Management Science* 17.3, pp. 219–229.

Harel, D. and Y. Koren (2004). "Graph drawing by high-dimensional embedding". In: *Journal of Graph Algorithms and Applications* 8.2, pp. 195–214.

Hastings, M. B. (2006). "Community detection as an inference problem". In: *Physical Review E* 74.3, p. 35102.

Herman, I., G. Melançon, and M. S. Marshall (2000). "Graph visualization and navigation in information visualization: A survey". In: *IEEE Transactions on Visualization and Computer Graphics* 6.1, pp. 24–43.

Hirt, M. R. et al. (2017). "A general scaling law reveals why the largest animals are not the fastest". In: *Nature Ecology & Evolution* 1.8, pp. 1116–1122.

Ho, H.-C. (2020). "The effects of foraging behaviour on food-web structure and dynamics". PhD thesis.

Ho, H. et al. (2019). "Predation risk influences food web structure by constraining species diet choice". In: *Ecology letters* 22.11, pp. 1734–1745.

Hodgkin, A. L. and A. F. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4, p. 500.

Holling, C. S. (1973). "Resilience and stability of ecological systems". In: *Annual Review of Ecology, Evolution, and Systematics* 4.1, pp. 1–23.

Holten, D. (2006). "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5, pp. 741–748.

Holten, D., P. Isenberg, et al. (2011). "An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs". In: *IEEE Pacific Visualization Symposium*. IEEE, pp. 195–202.

Holten, D. and J. J. van Wijk (2009). "Force-directed edge bundling for graph visualization". In: *Computer Graphics Forum* 28.3, pp. 983–990.

Hooke, R. (1678). *Explaining the power of springing bodies*, pp. 1–56.

Hu, Y. (2005). "Efficient, high-quality force-directed graph drawing". In: *Mathematica Journal* 10.1, pp. 37–71.

Hurter, C., O. Ersoy, and A. Telea (2012). "Graph bundling by kernel density estimation". In: *Computer Graphics Forum* 31.3pt1, pp. 865–874.

Hutchinson, G. E. (1961). "The paradox of the plankton". In: *The American Naturalist* 95.882, pp. 137–145.

Ingram, S. and T. Munzner (2012). "Glint: An MDS framework for costly distance functions". In: *SIGRAD Conference on Interactive Visual Analysis of Data*, pp. 29–38.

Jakobsen, T. (2001). "Advanced character physics". In: *Game Developers Conference*, pp. 1–16.

Jia, Y., M. Garland, and J. C. Hart (2011). "Social network clustering and visualization using hierarchical edge bundles". In: *Computer Graphics Forum* 30.8, pp. 2314–2327.

Johnson, B. and B. Shneiderman (1991). "Tree-maps: A space-filling approach to the visualization of hierarchical information structures". In: *IEEE Conference on Visualization*, pp. 284–291.

Johnson, D. B. (1975). "Finding all the elementary circuits of a directed graph". In: *SIAM Journal on Computing* 4.1, pp. 77–84.

Johnson, S., V. Domínguez-García, et al. (2014). "Trophic coherence determines food-web stability". In: *Proceedings of the National Academy of Sciences* 111.50, pp. 17923–17928.

Josuttis, N. M. (2012). "STL containers". In: *The C++ standard library: A tutorial and reference*. Addison-Wesley, pp. 253–392.

Kaiser, D. (2005). "Physics and Feynman's Diagrams". In: *American Scientist* 93.2, pp. 156–165.

– (2009). *Drawing theories apart: The dispersion of Feynman diagrams in postwar physics*. University of Chicago Press.

Kamada, T. and S. Kawai (1989). "An algorithm for drawing general undirected graphs". In: *Information Processing Letters* 31.1, pp. 7–15.

Khoury, M. et al. (2012). "Drawing large graphs by low-rank stress majorization". In: *Computer Graphics Forum* 31.3, pp. 975–984.

Kipf, T. N. and M. Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint*, pp. 1–14. arXiv: 1609.02907.

Kirmse, A. (2004). "Critically damped ease-in/ease-out smoothing". In: *Game programming gems*, pp. 90–102.

Kleiber, M. (1947). "Body size and metabolic rate". In: *Physiological Reviews* 27.4, pp. 511–541.

Knuth, D. E. (1993). *The Stanford GraphBase: A platform for combinatorial computing*. AcM Press New York.

Koebe, P. (1936). "Kontaktprobleme der konformen abbildung". In: *Ber. Sächs. Akad. Wiss. Leipzig* 88, pp. 141–164.

Koren, Y. (2003). "On spectral graph drawing". In: *International Computing and Combinatorics Conference*, pp. 496–508.

Kozlovsky, D. G. (1968). "A critical evaluation of the trophic level concept. I. Ecological efficiencies". In: *Ecology* 49.1, pp. 48–60.

Kruskal, J. B. (1964a). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". In: *Psychometrika* 29.1, pp. 1–27.

– (1964b). "Nonmetric multidimensional scaling: A numerical method". In: *Psychometrika* 29.2, pp. 115–129.

Krzywinski, M. et al. (2012). "Hive plots—rational approach to visualizing networks". In: *Briefings in Bioinformatics* 13.5, pp. 627–644.

Kullback, S. and R. A. Leibler (1951). "On information and sufficiency". In: *The Annals of Mathematical Statistics* 22.1, pp. 79–86.

Laird, S., D. Lawson, and H. J. Jensen (2008). "The tangled nature model of evolutionary ecology: An overview". In: *Mathematical Modeling of Biological Systems, Volume II*. Springer, pp. 49–62.

Lambert, A., R. Bourqui, and D. Auber (2010). "Winding roads: Routing edges into bundles". In: *Computer Graphics Forum*. Vol. 29. 3. Wiley Online Library, pp. 853–862.

Lamrous, S. and M. Taileb (2006). "Divisive hierarchical k-means". In: *International Conference on Computational Inteligence for Modelling Control and Automation*, pp. 1–6.

Lancichinetti, A., S. Fortunato, and F. Radicchi (2008). "Benchmark graphs for testing community detection algorithms". In: *Physical Review E* 78.4, p. 46110.

LeCun, Y. et al. (2017). "Geometric deep learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4, pp. 18–42.

Lee, A., D. Archambault, and M. Nacenta (2019). "Dynamic network plaid: A tool for the analysis of dynamic networks". In: *CHI conference on human factors in computing systems*. ACM, pp. 1–14.

Lee, J., W. Kladwang, et al. (2014). "RNA design rules from a massive open laboratory". In: *Proceedings of the National Academy of Sciences* 111.6, pp. 2122–2127.

Lhuillier, A., C. Hurter, and A. Telea (2017). "State of the art in edge and trail bundling techniques". In: *Computer Graphics Forum* 36.3, pp. 619–645.

Liiv, I. (2010). "Seriation and matrix reordering methods: An historical overview". In: *Statistical Analysis and Data Mining* 3.2, pp. 70–91.

Lindeman, R. L. (1942). "The trophic-dynamic aspect of ecology". In: *Ecology* 23.4, pp. 399–417.

Link, W. A., J. R. Sauer, and D. K. Niven (2008). "Combining breeding bird survey and Christmas Bird Count data to evaluate seasonal components of population change in northern bobwhite". In: *The Journal of Wildlife Management* 72.1, pp. 44–51.

Lusseau, D. et al. (2003). "The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations". In: *Behavioral Ecology and Sociobiology* 54.4, pp. 396–405.

Marchal, C. (2012). *The three-body problem*. Elsevier.

Masters, K. L. and Galaxy Zoo Team (2019). "Twelve years of Galaxy Zoo". In: *Proceedings of the International Astronomical Union* 14.S353, pp. 205–212.

May, R. M. (1973). *Stability and complexity in model ecosysytems*. Princeton University Press.

McDiarmid, C. and T. Müller (2014). "The number of disk graphs". In: *European Journal of Combinatorics* 35, pp. 413–431.

Mi, P. et al. (2016). "Interactive graph layout of a million nodes". In: *Informatics*. Vol. 3. 4. Multidisciplinary Digital Publishing Institute, pp. 1–19.

Minard, C. J. (1862). *Des tableaux graphiques et des cartes figuratives*, pp. 1–24.

Misue, K. et al. (1995). "Layout adjustment and the mental map". In: *Journal of Visual Languages & Computing* 6.2, pp. 183–210.

Nagy, K. A., I. A. Girard, and T. K. Brown (1999). "Energetics of free-ranging mammals, reptiles, and birds". In: *Annual Review of Nutrition* 19.1, pp. 247–277.

Newman, M. E. J. (2006a). "Finding community structure in networks using the eigenvectors of matrices". In: *Physical Review E* 74.3, p. 36104.

– (2006b). "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23, pp. 8577–8582.

Newman, M. E. J. and M. Girvan (2004). "Finding and evaluating community structure in networks". In: *Physical Review E* 69.2, p. 26113.

Noack, A. (2007). "Energy Models for Graph Clustering". In: *Journal of Graph Algorithms and Applications* 11.112, pp. 453–480.

Nocaj, A., M. Ortmann, and U. Brandes (2015). "Untangling the hairballs of multi-centered, small-world online social media networks". In: *Journal of Graph Algorithms and Applications* 19.2, pp. 595–618.

Ocubo, A. (1980). *Diffusion and ecological problems: Mathematical models*. Springer.

Okoe, M., R. Jianu, and S. Kobourov (2018). "Node-link or adjacency matrices: Old question, new insights". In: *IEEE Transactions on Visualization and Computer Graphics* 25.10, pp. 2940–2952.

Olauson, J., M. Marin, and L. Söder (2020). "Creating power system network layouts: A fast parallel algorithm". In: *IEEE Systems Journal* 14.3, pp. 3687–3694.

Ortmann, M., M. Klimenta, and U. Brandes (2017). "A sparse stress model". In: *Journal of Graph Algorithms and Applications* 21.5, pp. 791–821.

Page, L. et al. (1999). "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30, pp. 107–117.

Pawar, S. (2015). "The role of body size variation in community assembly". In: *Advances in Ecological Research* 52, pp. 201–248.

Pawar, S., A. I. Dell, and V. M. Savage (2012). "Dimensionality of consumer search space drives trophic interaction strengths". In: *Nature* 486.7404, pp. 485–489. arXiv: NIHMS150003.

Pearson, K. (1901). "On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine* 2.11, pp. 559–572.

Perozzi, B., R. Al-Rfou, and S. Skiena (2014). "Deepwalk: Online learning of social representations". In: *SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 701–710.

Petchey, O. L. et al. (2008). "Size, foraging, and food web structure". In: *Proceedings of the National Academy of Sciences* 105.11, pp. 4191–4196.

Pirlot, M. and R. V. V. Vidal (1996). "Simulated annealing: A tutorial". In: *Control and Cybernetics* 25, pp. 9–32.

Pons, P. and M. Latapy (2006). "Computing communities in large networks using random walks". In: *Journal of Graph Algorithms and Applications* 10.2, pp. 191–218.

Ponti, M. et al. (2018). "Getting it right or being top rank: Games in citizen science". In: *Citizen Science: Theory and Practice* 3.1.

Porter, M. M. and P. Niksiar (2018). "Multidimensional mechanics: Performance mapping of natural biological systems using permutated radar charts". In: *PLoS ONE* 13.9, e0204309.

Post, D. M. (2002). "The long and short of food-chain length". In: *Trends in Ecology & Evolution* 17.6, pp. 269–277.

Press, W. H. et al. (2007a). "Integration of ordinary differential equations". In: *Numerical recipes: The art of scientific computing*. 3rd ed., pp. 899–954.

– (2007b). "Minimization or maximization of functions". In: *Numerical recipes: The art of scientific computing*. 3rd ed. Cambridge University Press, pp. 487–562.

– (2007c). "Solution of linear algebralc equations". In: *Numerical recipes: The art of scientific computing*. 3rd ed. Cambridge University Press, pp. 37–109.

Pupyrev, S. et al. (2016). "Edge routing with ordered bundles". In: *Computational Geometry: Theory and Applications* 52, pp. 18–33.

Purchase, H. C. (1997). "Which aesthetic has the greatest effect on human understanding?" In: *International Symposium on Graph Drawing*. Springer, pp. 248–261.

– (2014). "A healthy critical attitude: Revisiting the results of a graph drawing study". In: *Journal of Graph Algorithms and Applications* 18.2, pp. 281–311.

Purchase, H. C., C. Pilcher, and B. Plimmer (2010). "Graph drawing aesthetics – created by users, not algorithms". In: *IEEE Transactions on Visualization and Computer Graphics* 18.1, pp. 81–92.

Qiu, H. and E. R. Hancock (2007). "Clustering and embedding using commute times". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.11, pp. 1873–1890.

Quinn, N. and M. Breuer (1979). "A forced directed component placement procedure for printed circuit boards". In: *IEEE Transactions on Circuits and Systems* 26.6, pp. 377–388.

Ramdas, A., N. G. Trillos, and M. Cuturi (2017). "On Wasserstein two-sample testing and related families of nonparametric tests". In: *Entropy* 19.2, p. 47.

Read, R. C. (1987). "A new method for drawing a planar graph given the cyclic order of the edges at each vertex". In: *Congressus Numerantium* 56, pp. 31–44.

Reichardt, J. and D. R. White (2007). "Role models for complex networks". In: *The European Physical Journal B* 60.2, pp. 217–224.

Robbins, H. and S. Monro (1951). "A stochastic approximation method". In: *The Annals of Mathematical Statistics*, pp. 400–407.

Rogowitz, B. E. and L. A. Treinish (1998). "Data visualization: The end of the rainbow". In: *IEEE Spectrum* 35.12, pp. 52–59.

Rosenzweig, M. L. (1971). "Paradox of enrichment: Destabilization of exploitation ecosystems in ecological time". In: *Science* 171.3969, pp. 385–387.

Rossberg, A. G. et al. (2006). "Food webs: Experts consuming families of experts". In: *Journal of Theoretical Biology* 241.3, pp. 552–563.

Rossiter, D. G. et al. (2015). "Can citizen science assist digital soil mapping?" In: *Geoderma* 259, pp. 71–80.

Rosvall, M. and C. T. Bergstrom (2008). "Maps of random walks on complex networks reveal community structure". In: *Proceedings of the National Academy of Sciences* 105.4, pp. 1118–1123.

Roux, M. (2018). "A comparative study of divisive and agglomerative hierarchical clustering algorithms". In: *Journal of Classification* 35.2, pp. 345–366.

Royer, L. et al. (2008). "Unraveling protein networks with power graph analysis". In: *PLoS Computational Biology* 4.7, e1000108.

Ruder, S. (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint*, pp. 1–14. arXiv: 1609.04747.

Rumpho, M. E. et al. (2008). "Horizontal gene transfer of the algal nuclear gene psbO to the photosynthetic sea slug Elysia chlorotica". In: *Proceedings of the National Academy of Sciences* 105.46, pp. 17867–17871.

Saffrey, P. and H. C. Purchase (2008). "The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study". In: *Conference on Australasian User Interface*. Citeseer, pp. 85–93.

Sankey, H. R. (1896). "The thermal efficiency of steam engines". In: *Minutes of the Proceedings of the Institution of Civil Engineers*. Vol. 125. 1896, pp. 182–212.

Savage, V. M. et al. (2004). "Effects of body size and temperature on population growth". In: *The American Naturalist* 163.3, pp. 429–441.

Schaefer, M., E. Sedgwick, and D. Štefankovič (2003). "Recognizing string graphs in NP". In: *Journal of Computer and System Sciences* 67.2, pp. 365–380.

Sederberg, T. W. (2005). *An introduction to B-spline curves*. Tech. rep. Brigham Young University, pp. 1–12.

Senior, A. W. et al. (2020). "Improved protein structure prediction using potentials from deep learning". In: *Nature* 577.7792, pp. 706–710.

Sibson, R. (1973). "SLINK: An optimally efficient algorithm for the single-link cluster method". In: *The Computer Journal* 16.1, pp. 30–34.

Sierpiński, W. (1915). "Sur une courbe dont tout point est un point de ramification". In: *Comptes rendus de l'Académie des Sciences* 160, pp. 302–305.

Simonetto, P. et al. (2011). "ImPrEd: An improved force directed algorithm that prevents nodes from crossing edges". In: *Computer Graphics Forum*. Vol. 30. 3. Wiley Online Library, pp. 1071–1080.

Smart, S., K. Wu, and D. A. Szafir (2019). "Color Crafting: Automating the Construction of Designer Quality Color Ramps". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1, pp. 1215–1225.

Smilkov, D. et al. (2019). "Tensorflow. js: Machine learning for the web and beyond". In: *arXiv preprint*. arXiv: 1901.05350.

Smith, D. W., R. O. Peterson, and D. B. Houston (2003). "Yellowstone after wolves". In: *BioScience* 53.4, pp. 330–340.

Sneath, P. H. A. and R. R. Sokal (1973). "Taxonomic structure". In: *Numerical taxonomy: The principles and practice of numerical classification*. Pp. 188–308.

Sommer, U. (1983). "Nutrient competition between phytoplankton species in multispecies chemostat experiments". In: *Archiv für Hydrobiologie* 96, pp. 399–416.

Stein, S. K. (1951). "Convex maps". In: *Proceedings of the American Mathematical Society* 2.3, pp. 464–466.

Steinitz, E. (1922). "Polyeder und raumeinteilungen". In: *Encyclopädie der Mathematischen Wissenschaften* 3, pp. 1–139.

Sugiyama, K. and K. Misue (1994). "A simple and unified method for drawing graphs: Magnetic-spring algorithm". In: *International Symposium on Graph Drawing*. Springer, pp. 364–375.

Suh, A. et al. (2019). "Persistent homology guided force-directed graph layouts". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1, pp. 697–707.

Tang, S., S. Pawar, and S. Allesina (2014). "Correlation between interaction strengths drives stability in large ecological networks". In: *Ecology Letters* 17.9, pp. 1094–1100.

Telea, A. (2018). "Image-based graph visualization: Advances and challenges". In: *International Symposium on Graph Drawing*. Springer, pp. 3–19.

Traag, V. A., L. Waltman, and N. J. van Eck (2019). "From Louvain to Leiden: Guaranteeing well-connected communities". In: *Scientific Reports* 9.5233, pp. 1–12.

Traud, A. L., P. J. Mucha, and M. A. Porter (2012). "Social structure of Facebook networks". In: *Physica A: Statistical Mechanics and its Applications* 391.16, pp. 4165–4180.

Tutte, W. T. (1963). "How to draw a graph". In: *Proceedings of the London Mathematical Society* 3.1, pp. 743–767.

Villani, C. (2003). "The metric side of optimal transportation". In: *Topics in Optimal Transportation*, pp. 205–236.

Virtanen, P. et al. (2020). "SciPy 1.0: Fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3, pp. 261–272.

Vogogias, A. et al. (2016). "MLCut: Exploring multi-level cuts in dendrograms for biological data". In: *Computer Graphics and Visual Computing Conference*, pp. 1–8.

Von Landesberger, T. et al. (2011). "Visual analysis of large graphs: State of the art and future research challenges". In: *Computer graphics forum*. Vol. 30. 6. Wiley Online Library, pp. 1719–1749.

Wagner, K. (1936). "Bemerkungen zum vierfarbenproblem". In: *Jber. Deutsches Mathematiks Verein* 46, pp. 26–32.

Walmsley, M. et al. (2020). "Galaxy Zoo: Probabilistic morphology through Bayesian CNNs and active learning". In: *Monthly Notices of the Royal Astronomical Society* 491.2, pp. 1554–1574.

Wang, W., H. Wang, et al. (2006). "Visualization of large hierarchical data by circle packing". In: *SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA: ACM Press, pp. 517–520.

Wang, Y. and Z. Wang (2012). "A fast successive over-relaxation algorithm for force-directed network graph drawing". In: *Science China Information Sciences* 55.3, pp. 677–688.

Ward Jr, J. H. (1963). "Hierarchical grouping to optimize an objective function". In: *Journal of the American Statistical Association* 58.301, pp. 236–244.

Watteaux, R., R. Stocker, and J. R. Taylor (2015). "Sensitivity of the rate of nutrient uptake by chemotactic bacteria to physical and biological parameters in a turbulent environment". In: *Journal of Theoretical Biology* 387, pp. 120–135.

Welling, M. and Y. W. Teh (2011). "Bayesian learning via stochastic gradient Langevin dynamics". In: *The International Conference on Machine Learning*, pp. 681–688.

West, G. B., J. H. Brown, and B. J. Enquist (1997). "A general model for the origin of allometric scaling laws in biology". In: *Science* 276.5309, pp. 122–126.

Williams, R. J. and N. D. Martinez (2000). "Simple rules yield complex foodwebs". In: *Nature* 404.6774, pp. 180–183.

– (2004). "Limits to trophic levels and omnivory in complex food webs: theory and data". In: *The American Naturalist* 163.3, pp. 458–468.

Wu, Z. et al. (2020). "A comprehensive survey on graph neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21.

Yen, L. et al. (2005). "Clustering using a random walk based distance measure". In: *European Symposium on Artificial Neural Networks*, pp. 27–29.

Yoghourdjian, V. et al. (2018). "Graph thumbnails: Identifying and comparing multiple graphs at a glance". In: *IEEE Transactions on Visualization and Computer Graphics* 24.12, pp. 3081–3095.

Young, D. M. (2014). *Iterative solution of large linear systems*. Elsevier.

Yuan, X. et al. (2012). "Intelligent graph layout using many users' input". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12, pp. 2699–2708.

Zachary, W. W. (1977). "An information flow model for conflict and fission in small groups". In: *Journal of Anthropological Research* 33.4, pp. 452–473.

Zhang, H. and X. He (2005). "Canonical ordering trees and their applications in graph drawing". In: *Discrete & Computational Geometry* 33.2, pp. 321–344.

Zheng, J. X., S. Pawar, and D. F. M. Goodman (2018). "Confluent* drawings by hierarchical clustering". In: *International Symposium on Graph Drawing*. Springer, pp. 640–642.

– (2019a). "EcoBuilder: A video game to crowdsource food web research". In: *British Ecological Society Annual Meeting*.

– (2019b). "Further towards unambiguous edge bundling: Investigating power-confluent drawings for network visualization". In: *IEEE Transactions on Visualization and Computer Graphics*.

– (2019c). "Graph drawing by stochastic gradient descent". In: *IEEE Transactions on Visualization and Computer Graphics* 25.9, pp. 2738–2748.