Algorithms for the Analysis and Visualization of Biomedical Networks

Vassilis L. Tsiaras

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Supervisor: Professor Ioannis G. Tollis

Computer Science Department
University of Crete
Heraklion

October 2009

Abstract

Networks are prevalent in life sciences. Two characteristic examples are Gene Ontology and brain functional networks.

Gene Ontology (GO) is structured as a Directed Acyclic Graph (DAG). Motivated by the fact that visualizing the whole GO with node-link representation leads to an unintelligible image, we propose space filling visualization for DAGs. Space filling visualizations, such as the treemaps, have the capacity to display thousands of items legibly in limited space. Treemaps have been used to visualize the GO by first transforming the DAG into a tree. However this transformation has several undesirable effects such as producing trees with a large number of nodes and scattering the rectangles associated with the duplicates of a node around the display rectangle. In this thesis we introduce the problem of visualizing a DAG with space filling techniques without converting it into a tree first. We define drawing constraints that generalize treemaps to space filling visualizations of DAGs which we call *DAGmaps*. Then we show that deciding whether or not a DAG admits a DAGmap drawing is NP-complete. For this reason we study two special cases of the problem and we propose a heuristic algorithm that reduces the vertex duplications. First, we define a special case of the problem called one-dimensional DAGmap where the initial rectangle is sliced in one-dimension (e.g. the vertical). We prove that a DAG admits a one-dimensional DAGmap if and only if it admits a directed ε-visibility representation. This one-to-one correspondence between the two problems leads to characterization of the class of DAGs that admit a one-dimensional DAGmap as well as to elegant linear time decision and drawing algorithms. Another special case of the problem occurs when the input is restricted to Two Terminal Series Parallel (TTSP) digraphs. We show that every TTSP digraph admits a DAGmap which can be drawn using the decomposition tree of G in linear time. The heuristic algorithm that we propose decomposes a DAG into component st-graphs using dominance relationships and then combines the drawings of the component st-graphs. In case that an st-graph does not admit a DAGmap then it is transformed into a TTSP digraph via vertex duplications. This algorithm performs fewer vertex duplications than the transformation of a DAG into a tree when the DAG can be decomposed into component st-graphs. Finally we implemented all the proposed algorithms in a program called DAGmap View. This program, which also implements the novel feature of separate layout and nesting functions, is an ideal tool to visualize and navigate through GO.

Brain functional networks are modeled with valued graphs where the vertices correspond to brain areas and the edges denote statistical dependence between brain areas. The edge values belong to the interval $(0, 1]$ and are interpreted as strength of dependence. We call these networks *greyscale* since they can be visualized with different shades of grey. Motivated by the fact that a program that analyzes and visualizes brain functional networks is of interest to researchers working in brain connectivity, we created BrainNetVis. This program displays greyscale networks and implements a number of measures, most of which were selected from the literature. However we also contributed to the development of the theory of greyscale networks by proposing: a) a function that converts edge strength into edge lengths; b) a formula that calculates the

importance of a vertex due to its position on a network as well as due to its attributes and c) some generalizations of graph theoretic measures to greyscale networks. We demonstrate BrainNetVis through two case studies. The first case study compares brain functional networks of alcoholic and control subjects during memory rehearsal tasks and shows that there are statistically significant differences between the two groups at beta band. The second case study compares brain functional networks of a healthy subject during two motor imagery tasks (left hand and foot). We found that the corresponding networks are very similar and we propose a method to differentiate between the two imagery tasks using network statistics.

Περίληψη

Τα δίκτυα χρησιμοποιούνται ευρύτατα για την μοντελοποίηση γνώσης και δεδομένων στον τομέα των βιοεπιστημών. Δύο χαρακτηριστικά παραδείγματα είναι η οντολογία γονιδίων και τα λειτουργικά δίκτυα του εγκεφάλου.

Η οντολογία γονιδίων (GO) είναι δομημένη ως κατευθυνόμενος άκυκλος γράφος (DAG). Η οπτικοποίηση ολόκληρης της οντολογίας γονιδίων, με τους κόμβους να αναπαρίστανται ως μικροί κύκλοι ή πολύγωνα και τις ακμές ως γραμμές, οδηγεί σε μια δυσνόητη εικόνα. Γι΄ αυτό προτείνουμε την τεχνική της οπτικοποίησης με πλήρωση του χώρου για κατευθυνόμενους άκυκλους γράφους. Αυτή η τεχνική έχει δοκιμαστεί σε δένδρα (treemaps) και παρέχει την δυνατότητα να απεικονιστούν ευανάγνωστα χιλιάδες κόμβοι σε περιορισμένο χώρο. Η τεχνική treemap έχει χρησιμοποιηθεί για να απεικονίσει την οντολογία γονιδίων, η οποία πρώτα μετατρέπεται σε δένδρο δημιουργώντας τόσα αντίγραφα ενός κόμβου v, όσα είναι και τα μονοπάτια που ξεκινούν από τον κόμβο αφετηρία και καταλήγουν στον κόμβο v. Όμως αυτή η μετατροπή δημιουργεί σημαντικά προβλήματα, όπως ο μεγάλος αριθμός κόμβων των παραγόμενων δένδρων και το ότι τα αντίγραφα ενός κόμβου βρίσκονται διασκορπισμένα μέσα στην περιοχή σχεδίασης. Σε αυτή την διατριβή παρουσιάζεται το πρόβλημα της απεικόνισης κατευθυνόμενων άκυκλων γράφων με πλήρωση του χώρου, χωρίς ο γράφος να μετατραπεί σε δένδρο. Καταρχήν, ορίζεται το πρόβλημα έτσι, ώστε να γενικεύει την περίπτωση των δένδρων και ονομάζεται DAGmap. Κατόπιν, αποδεικνύεται ότι το να αποφασίσουμε αν ένας τυχαίος γράφος επιδέχεται μια απεικόνιση DAGmap είναι ένα NP-πλήρες πρόβλημα. Επομένως, αξίζει να μελετηθούν ειδικές περιπτώσεις του προβλήματος, και ευρετικοί αλγορίθμοι που μετατρέπουν τον γράφο έτσι, ώστε να επιδέχεται μια απεικόνιση DAGmap. Μια ειδική περίπτωση του προβλήματος ονομάζεται μονοδιάστατο DAGmap, διότι το αρχικό ορθογώνιο χωρίζεται κατά μήκος μίας μόνο διάστασης, π.χ., την κάθετη. Αποδεικνύεται ότι ένας κατευθυνόμενος άκυκλος γράφος επιδέχεται ένα μονοδιάστατο DAGmap, εάν και μόνο εάν ο γράφος επιδέχεται μια αναπαράσταση κατευθυνόμενης ε-ορατότητας. Αυτή η ένα-προς-ένα αντιστοιχία των δύο προβλημάτων οδηγεί σε πλήρη χαρακτηρισμό της κατηγορίας των γράφων που επιδέχονται μονοδιάστατο DAGmap, καθώς και σε γραμμικούς αλγορίθμους απόφασης και σχεδίασης. Μια άλλη ειδική περίπτωση του προβλήματος εμφανίζεται όταν οι γράφοι είναι κατευθυνόμενοι σειριακοί-παράλληλοι δύο τερματικών (TTSP). Αποδεικνύεται ότι κάθε TTSP γράφος επιδέχεται απεικόνιση DAGmap, η οποία μπορεί να ζωγραφιστεί σε γραμμικό χρόνο χρησιμοποιώντας το δένδρο αποδόμησης του γράφου. Ο ευρετικός αλγόριθμος βασίζεται σε σχέσεις κυριαρχίας μεταξύ των κόμβων και χωρίζει ένα κατευθυνόμενο άκυκλο γράφο σε συνιστώσες έτσι, ώστε η κάθε μία από αυτές να συνδέεται με τον υπόλοιπο γράφο μόνο μέσω του κόμβου αφετηρίας και/ή τερματισμού της. Έπειτα ο αλγόριθμος συνδυάζει τις απεικονίσεις των συνιστωσών για να βρει την απεικόνιση του γράφου. Σε περίπτωση που μια συνιστώσα δεν επιδέχεται DAGmap, τότε μετατρέπεται σε TTSP γράφο. Οι αλγόριθμοι των δύο ειδικών περιπτώσεων του προβλήματος και ο ευρετικός αλγόριθμος έχουν υλοποιηθεί στο πρόγραμμα DAGmap View. Το πρόγραμμα αυτό έχει σχεδιαστεί με αρκετές νέες ιδέες, όπως ο διαχωρισμός της συνάρτησης εύρεσης συντεταγμένων των ορθογωνίων από την συνάρτηση παρουσίασης της δομής της ιεραρχίας του γράφου. Επίσης, το DAGmap View είναι ένα ιδανικό εργαλείο για την οπτικοποίηση και την πλοήγηση στην οντολογία γονιδίων.

Τα λειτουργικά δίκτυα του εγκεφάλου μοντελοποιούνται από γράφους με τιμές στις ακμές τους. Οι κόμβοι αντιστοιχούν σε περιοχές του εγκεφάλου και οι ακμές φανερώνουν στατιστική εξάρτηση μεταξύ των αντίστοιχων περιοχών. Οι τιμές των ακμών ανήκουν στο διάστημα (0, 1] και θεωρούνται ως ένταση της εξάρτησης. Τέτοιου είδους δίκτυα ονομάζονται γκρίζας κλίμακας (greyscale), διότι μπορούν να οπτικοποιηθούν με αποχρώσεις του γκρί. Δεδομένου ότι δεν υπάρχει διαθέσιμο ένα πρόγραμμα που αναλύει και οπτικοποιεί greyscale δίκτυα, ενώ είναι χρήσιμο στους ερευνητές που ασχολούνται με την συνδεσιμότητα του εγκεφάλου, δημιουργήσαμε το BrainNetVis. Το πρόγραμμα αυτό οπτικοποιεί τα greyscale δίκτυα και υπολογίζει μια σειρά από μετρικές δικτύων, οι περισσότερες από τις οποίες επιλέχθηκαν από τη σχετική βιβλιογραφία. Επίσης παρουσιάζονται και νέα θεωρητικά αποτελέσματα: α) μια συνάρτηση που μετατρέπει την ένταση μιας ακμής σε μήκος της ακμής, β) ένας τύπος που υπολογίζει την σημασία ενός κόμβου, τόσο με βάση την θέση του στο δίκτυο, όσο και από τα χαρακτηριστικά του, και γ) ορισμένες γενικεύσεις για greyscale δίκτυα μετρικών που προϋπήρχαν για γράφους. Τέλος, η παρουσίαση του προγράμματος BrainNetVis γίνεται μέσω δύο αναλύσεων δεδομένων εγκεφαλογραφημάτων. Στην πρώτη ανάλυση συγκρίνονται τα λειτουργικά δίκτυα του εγκεφάλου αλκοολικών και μη αλκοολικών ατόμων κατά την διάρκεια ενός πειράματος προσωρινής μνήμης. Αποδεικνύεται ότι υπάρχουν στατιστικά σημαντικές διαφορές μεταξύ των δύο ομάδων στη βήτα ζώνη συχνοτήτων (beta band). Στην δεύτερη μελέτη συγκρίνονται τα λειτουργικά δίκτυα του εγκεφάλου ενός υγιούς ατόμου κατά την διάρκεια εικονικών κινήσεων είτε του ποδιού του (οποιοδήποτε από τα δύο) είτε του αριστερού χεριού του. Διαπιστώνεται ότι τα αντίστοιχα δίκτυα είναι σχεδόν όμοια και προτείνεται μια μέθοδος για την διάκριση μεταξύ των δύο περιπτώσεων (ποδιού - αριστερού χεριού).

Acknowledgements

First of all, I wish to express my gratitude towards my advisor, Professor Ioannis Tollis, who guided and supported me during this thesis. His office door was always open to me whenever I needed his advice. I also want to thank him for suggesting me the DAGMAP problem and for his excellent brainstorming, such as his idea that a two terminal series parallel digraph might admit a DAGmap and his suggestions concerning ε-visibility.

I would also like to thank the members of my dissertation committee. Many thanks to Vassilis Christophides for his comments, which were of great value for the improvement of the quality of this thesis. Many thanks to Panagiotis Tsakalides and Ioannis Tsamardinos for their comments and suggestions on my final presentation. Many thanks to Panagiotis Tsakalides for teaching me autoregressive models for time series. Many thanks to Menelaos Karavelas for our fruitful discussions on greyscale and complex networks. Many thanks to Martin Reczko for his suggestions on the design of the DAGmap View program. Many thanks also to Panayiotis Benos for his willingness to check my dissertation from the point of view of biologists. Many thanks to Michalis Zervakis for his advices and for traveling to Heraklion to participate in my thesis defence.

I would also like to express my gratitude to all my collaborators: Dimitris Andreou, Sofia Triantafillou, Vangelis Sakkalis and Michalis Zervakis. Thank you for the hours we spent thinking together.

I also want to thank all my other teachers at University of Crete. I thank Ioannis Tollis, Georgios Georgakopoulos, George Tziritas, Dimitris Tsakiris, Panagiotis Tsakalides, Yannis Stylianou, Antonis Argyros, Panos Trahanias, Yannis Manoussakis and Nikos Komodakis. Especially, I thank Evangelos Markatos for the excellent course on parallel systems and grids. This course inspired me to apply for a Ph.D. degree in computer science.

I am indebted to the secretary of post-graduate studies Rena Kalaitzaki for her kindness, because she encouraged me to resubmit an application for a Ph.D. degree and because she responded quickly in all my questions and applications. Her quick response was crucial for taking paid educational leave from my service as a teacher in secondary schools. Here I should mention that, this Ph.D. has been funded by the Greek Ministry of Education through my paid educational leave from my duties.

Last but not least, I wish to thank my wife Anastasia for her wholehearted support. She suffered more than me during all these years and for this reason this thesis is dedicated to her.

Contents

1 Introduction **12**

2 Space Filling Visualization of Directed Acyclic Graphs **17**
2.1 Motivation: Visualizing the Gene Ontology 17
2.2 Preliminaries: Elements of Graph Theory 19
2.3 Problem Definition 21
2.3.1 Drawing Constraints 21
2.3.2 Examples and Counter-Examples of DAGs that Admit a DAGmap 23
2.3.3 Vertex Duplication 24
2.3.4 Exponential Increase in the Number of Vertices 24
2.4 Treemap Algorithms 25
2.4.1 Treemap Generic Algorithm 26
2.4.2 Slice and Dice Algorithm 26
2.4.3 Squarified Algorithm 26
2.5 Special Cases 28
2.5.1 Two Terminal Series Parallel Digraphs 28
2.5.2 Minimal Vertex Series Parallel Digraphs 35
2.5.3 One-Dimensional DAGmaps 37
2.5.4 Minimization of Vertex Duplications in One-Dimensional DAGmaps 41
2.6 The General Case 42
2.6.1 The Recognition Problem 42
2.6.2 Minimization of Vertex Duplication 47

3 DAGmaps and Directed ε-Visibility Representations **49**
3.1 Preliminaries 50
3.2 One-Dimensional DAGmaps and Directed ε-Visibility Representations 50
3.3 Characterization of Directed ε-Visibility Representations 52
3.4 Characterization of One-Dimensional DAGmaps 56
3.5 DAGmaps and Directed Three-Dimensional ε-Visibility Representations 58

4 DAGmaps and Dominance Relationships **60**
4.1 Preliminaries 60
4.2 Dominance and DAGmap Visualization 61
4.2.1 Separating the subgraphs of a DAG 64

4.2.2 Duplication Algorithm 64
4.2.3 DAGmap Algorithm 66
4.2.4 Finding Dominators 66

5 Technical Issues and Software for DAGmaps 69
5.1 Hierarchy Presentation 69
5.2 Nesting 70
5.2.1 Nesting in Treemaps 70
5.2.2 Nesting in TTSP digraphs 74
5.3 DAGmap View: A Tool for Space Filling Visualization of DAGs . 78
5.4 Description 78
5.5 Recognized Classes of DAGs 83
5.6 Duplication Heuristics 85
5.7 Discussion 85

6 Brain Functional Connectivity Estimation from EEG 88
6.1 EEG origin 89
6.2 EEG Oscillatory Patterns 90
6.3 EEG Limitations and Artifacts 90
6.4 Common Spatial Patterns 91
6.5 Synchronization Measures 93
6.5.1 Preliminaries 93
6.5.2 Synchronization of Weakly Coupled Systems 93
6.5.3 Nonlinear Measures 94
6.5.4 Linear Measures 96
6.6 Thresholding 101
6.6.1 Choosing a threshold value θ 102

7 Analysis of Greyscale Networks 105
7.1 Preliminaries: Elements of Algebraic Graph Theory 105
7.2 Valued Networks and Real Matrices 106
7.3 Greyscale Networks 107
7.3.1 Edge Lengths in Greyscale Networks 107
7.3.2 Distances in Greyscale Networks 110
7.4 Vertex Importance Score 114
7.5 Centrality Measures 115
7.5.1 Centrality Measures Based on Neighborhoods 115
7.5.2 Centrality Measures Based on Distances 116
7.5.3 Spectral Centrality Measures 118
7.6 Network Level Measures 124
7.6.1 Average Degree 124
7.6.2 Network Centrality 124
7.6.3 Clustering Coefficient and Transitivity 125
7.6.4 Assortative Mixing 127

8 Analysis and Visualization of Greyscale Brain Functional Networks 130
8.1 BrainNetVis 130
8.2 Case Study: Alcoholic and Control Subjects 133
8.2.1 Data acquisition and analysis 133

8.2.2 Experimental setting and test description 134
8.2.3 Network construction 134
8.2.4 Results 134
8.3 Case Study: Motor Imagery 136
8.3.1 Experimental Setup 136
8.3.2 Results 138

9 Conclusions & Future Work **141**

10 Appendices **158**
10.1 Appendix I: Eigenvalue Computation 158
10.2 Appendix II: Squarified Layout Algorithm 158

List of Figures

2.1 A small subgraph of the GO DAG. 17
2.2 Node-link representation of the whole GO. GO consists of three three structured and controlled vocabularies: "biological process", "molecular function" and "cellular component". These vocabularies are shown in this order from left to right. 18
2.3 Transforming the GO DAG into a tree and then drawing it as a treemap. The first five layers from the root are shown. The tree structure is visualized via nesting. The multiple copies of the node "regulation of cellular process" are shown with green background color. The color of nodes refers to the relationship between a GO term and its parent GO term. We use white color for *is_a* relationship, light brown color for *part_of*, gray color for *regulates*, red color for *positively_regulates* and blue color for *negatively_regulates*. E.g., "metabolic process" *is_a* "biological process" which *is_a* "all". Also "cell part" *is_a* "cellular component" and *part_of* "cell". 19
2.4 An example where an edge rectangle is the intersection of the origin and destination vertex rectangles. In the case of multigraphs, if there are k parallel edges between an origin vertex and a destination vertex, the intersection rectangle is arbitrarily partitioned into k equal area rectangles. 23
2.5 In this example a subgraph of the GO DAG is drawn. The color refers to the relationship between two GO terms. We use white color for *is_a* relationship and light brown color for *part_of* relationship. The term "cellular metabolic process" *is_a* "cellular process" and *is_a* "metabolic process". The term "cell part" *is_a* "cellular component" and *part_of* "cell". (Compare the visualization of "cell part" between this figure and Fig. 2.3.) 23
2.6 Examples of DAGs that admit DAGmaps. 24
2.7 Counter-example of a DAG that does not admit a drawing that satisfy all constraints B1-B5. However by relaxing constraint B1 the DAG admits a DAGmap. 25
2.8 Counter-example of a DAG that does not admit a drawing that satisfy constraints B2-B5 if the vertices are drawn as simply connected regions of the plane. 25
2.9 After the duplication of vertex h the DAG of Fig. 2.7(a) is transformed into a DAG that admits a DAGmap. 25
2.10 Example of transforming a DAG into a tree and then drawing it as a treemap. 26

2.11 A DAG with four vertices, having the structure described in the worst case scenario. 26
2.12 Recursive definition of a TTSP digraph. 28
2.13 Given a TTSP digraph G, we construct its decomposition tree by performing series and parallel reductions. Then we merge the adjacent P-nodes. 30
2.14 Series and parallel reduction rules. 30
2.15 Example of a TTSP digraph DAGmap drawing. Only edge rectangles are shown. The hierarchy structure is shown via nesting. For an example where only vertex rectangles are shown see Fig. 2.16(i). 31
2.16 Step by step drawing of the DAG of Fig. 2.15. The vertex rectangles are shown with white color while the edge rectangles are shown with light gray color. The nesting algorithm, used in this example, is slightly different than the nesting algorithm used in Fig. 2.15(c), since care was taken to draw edge rectangles within the intersection of their origin and destination vertex rectangles. 33
2.17 Example of finding the inverse line digraph of a MVSP digraph . 36
2.18 Space filling visualization of the MVSP digraph of Fig. 2.17 . . . 37
2.19 An example of a one-dimensional DAGmap drawing of a DAG G. The hierarchy structure is visualized via nesting in the vertical direction. 38
2.20 A one-dimensional DAGmap example. 38
2.21 To each $\alpha_i \in A$ we correspond a DAG G_{α_i} with $\alpha_i + 1$ vertices and to G_{α_i} we correspond a drawing rectangle R_{α_i} of area α_i. . . 44
2.22 The drawing of the enforcer leaves m gaps, each of area Σ. 44
2.23 The DAG used as enforcer in the proof. 45
2.24 One possible drawing of the enforcer. 45
2.25 An example of how three rectangles, of total area Σ, fill a gap. . 46
2.26 The enforcer of the reduction in Theorem 2.8. 48

3.1 Directed ε-visibility representation of the DAG of Fig. 2.19(a) . . 51
3.2 Example of a run of Algorithm 8. Planar st-graphs G and G^* are labeled by topological numbering Y and X, respectively. 55
3.3 Steps 4 and 5 of Algorithm 9 when applied to DAG of Fig. 3.2(a) 57
3.4 A tree T, a treemap of T and the three-dimensional ε-visibility representation of T that is constructed from the treemap of T . . 59
3.5 The DAG of this figure does not admit a DAGmap 2.7(a). However it admits a three-dimensional directed ε-visibility representation. 59

4.1 (a) A DAG which can be decomposed into two component st-Graphs. (b) DAGmap of the component st-Graph which is induced by the vertices 1, 2, 3, 4, 5 and 6. This subgraph is a MVSP digraph 2.5.2 and admits a DAGmap. (c) DAGmap of the component st-Graph which is induced by the vertices 6, 7, 8 and 9. This subgraph is a TTSP digraph. (d) DAGmap of G which is a combination of DAGmaps of (b) and (c). 61
4.2 An example showing a DAG and its dominator and post-dominator trees. 62

4.3 A counter-example that shows that the converse of Theorem 4.1 does not hold. From the DAG and its corresponding DAGmap we see that $R_4 \subset R_1$ but vertex 1 does not dominate vertex 4. . . 62
4.4 An example of the transformation of a DAG G into G' that admits a DAGmap. a) A DAG with an artificial sink vertex t. b) Introduction of artificial b-vertices. c) Folding of st-subgraphs into c-vertices. d) Vertex duplications. e) Unfolding of c-vertices; the component st-graph $G_{6,11}$ appears twice in DAG G'. f) DAGmap drawing of G'. 63
4.5 DAG G of Fig. 4.2(a) is converted into a TTSP digraph G' by duplications of vertices 3, 4 and 5 and then a DAGmap of G' is drawn. a) Only the first three layers from the source are shown. b) The whole hierarchy is shown. 66

5.1 A path P in a tree and the rectangles that correspond to nodes of P. 71
5.2 Rectangle R_α is partitioned into rectangles R_β, R_γ and R_δ. Then rectangle $R_\beta \cup R_\gamma \cup R_\delta$ is shrunk to reveal a border of rectangle R_α. The distance of each side of the shrunk $R_\beta \cup R_\gamma \cup R_\delta$ from the corresponding side of R_α is d. 71
5.3 An example of how the nesting algorithm behaves on S-nodes of the decomposition tree T. 75
5.4 An example of how the nesting algorithm behaves on P-nodes of the decomposition tree T. 75
5.5 After the post processing step the rectangle of edge $(2, 5)$ is covered by the rectangle of vertex 5. 76
5.6 An example of the construction of graph G'. Each node of G is represented by an S-node in the decomposition tree T' of G'. . . 76
5.7 DAGmap View demo: The GO is converted into a tree and visualized as treemap. The subtree rooted at "regulation of metabolic process" is shown. The rectangle that correspond to term "regulation of RNA metabolic process" is selected. The selected term and its duplicates are painted green. The color of duplicates is slightly darker than that of the selected term. 80
5.8 DAGmap View demo: The options window. 81
5.9 DAGmap View demo: Some attributes of selected term "regulation of RNA metabolic process". 81
5.10 DAGmap View demo. 82
5.11 DAGmap View demo: The user changes the nesting borders on-line. 82
5.12 DAGmap View demo: The GO is converted into a tree and visualized as a treemap. 83
5.13 DAGmap View navigation demo: When a user double clicks on the border of "cellular metabolic process" of Fig. 5.12 the subtree rooted into this node is displayed. 84
5.14 DAGmap View demo: Zoom out followed by move left and move down of the image. 84

5.15 (a) and (b) Patterns of vertices that DAGmap View can draw without duplications. (c) The three terms: cellular component, cell and cell part as well as the three terms: cellular component, organelle and organelle part form patterns similar to the one of Fig. (a). These terms are drawn without duplication. Compare with Fig. (d) where the same terms are drawn with duplications. With green color are displayed the selected terms and their duplicates. (e) and (f) Drawing without and with duplications of the four terms: biological process, metabolic process, cellular process and cellular metabolic process. These terms form a pattern similar to the one of Fig. (b). 86

6.1 An overview of neurophysiological data processing and how network concepts can be used to extract information from data. . . 89
6.2 Examples of EEG artifacts. a) Alternating Current (AC) line noise shown by taking the power spectrum of a channel. The AC line operates at 50 Hz. b) Applying ICA to BCI data. We see that component 5 corresponds to eye blinks. 92
6.3 An example of how $R_n^{(k)}(Z)$ and $R_n^{(k)}(Z/Y)$ are calculated. The phase spaces of the two dynamical systems are denoted by Y and Z. The points represent m-dimensional vectors. The parameter k is fixed to 3. The calculation of $R_n^{(k)}(Z)$ is based on the three nearest neighbors of point $\mathbf{z}(n)$ in phase space Z which are shown by red arrows. The calculation of $R_n^{(k)}(Z/Y)$ is based on the three points in phase space Z that correspond to the three nearest neighbors of $\mathbf{y}(n)$ in phase space Y which are shown by blue arrows. For simplicity the Theiler correction is not considered. . 96
6.4 Example of threshold functions 102
6.5 Example of how the network statistics K and L differ between the alcoholic (black dashed) and control (blue) patients as the threshold varies. The networks were constructed by estimating the synchronization between the channel signals using the robust interdependence measure proposed by Arnhold et al. [18] 104

7.1 In greyscale networks the edge values can be visualized using different shades of grey. 107
7.2 Example of non-increasing functions that are used to transform the strength of a dependence into length between pairs of vertices 109
7.3 An example of counter-intuitive behavior when function $g(x) = M(1-x)$ is used to transform interdependence into distance. The path $a \to d$ has smaller distance than the path $a \to b \to c \to d$ and shortest-paths algorithms chose it for information transmission. However there is no direct interdependence between vertices a and d and the information between vertices a and d passes through vertices b and c. 109
7.4 Edge strength is converted into edge length using the inverse function $g(x) = 1/x$. Then from the edge strengths we calculate the importance and from edge lengths the distance of a path. . . 111

7.5 Shortest-path betweenness centrality in valued networks. Vertices 1, 2, 3, 4, 9, 10, and 11 have zero shortest-path betweenness centrality in both networks. In network A, vertices 5, 8, 6, and 7 have centralities 0.544, 0.478, 0.222, and 0.222 respectively. In network B, vertices 5, 8, 6, and 7 have centralities 0.533, 0.467, 0.422, and 0 respectively. 118
7.6 The second generalization of shortest-path betweenness centrality in valued networks. Vertices 2, 3, and 4 have zero shortest-path betweenness centrality as expected. However vertex 1 has also zero betweenness contrary to the fact that is the center of the network. If we apply a cutoff function to remove the edges with week values then the normalized centrality of vertex 1 is one. . . 119
7.7 Star network with 5 vertices. The largest eigenvalue of the adjacency matrix is $\lambda_1 = 2$. If we choose $\alpha = 0.55 > 1/\lambda_1$ then $\mathbf{c} = [-15.2381, -7.3810, -7.3810, -7.3810, -7.3810]^T$. If we choose $\alpha = 0.45 < 1/\lambda_1$ then $\mathbf{c} = [14.7368, 7.6316, 7.6316, 7.6316, 7.6316]^T$ which makes more sense. 122
7.8 The above network has $n+2$ vertices in total: Vertices $1, 2, \ldots, n$, plus other two vertices a and b. The transitivity for this network is $T(G) = \frac{3n}{\binom{n+1}{2}+\binom{n+1}{2}+1+\ldots+1} = \frac{3}{n+2}$, which converges to 0 as $n \to \infty$. On the other hand, the clustering coefficient is $C(G) = \frac{1}{n+2}(\frac{2}{n+1} + \frac{2}{n+1} + 1 + \ldots + 1) = \frac{n^2+n+4}{n^2+3n+2}$, which converges to 1 when $n \to \infty$ 126
7.9 An example of how to use (7.56) for undirected graphs. Each undirected edge is substituted with two directed edges pointing in opposite directions. The set of edges is: $E = \{(1,2), (2,1), (2,3), (3,2), (2,4), (4,2), (3,4), (4,3)\}$. Suppose that we chose function ρ to be the degree of the directed graph ($\rho \equiv \deg$). Then the list of degrees that correspond to origin vertices of edges in E is: $x = [2, 6, 6, 4, 6, 4, 4, 4]$. Similarly the list of degrees that correspond to destination vertices of edges in E is: $y = [6, 2, 4, 6, 4, 6, 4, 4]$. Then the assortativity coefficient is $r = \mathrm{corr}(x, y) = -0.714286$ 129

8.1 A view of BrainNetVis program. 131
8.2 Visualization of network of Fig. 8.1 using multidimensional scaling (left) and binary stress (right). 132
8.3 Vertex level (centrality) measures of network of Fig. 8.1. 132
8.4 Spectral power density topographic map. See Fig. 8.9 for more details. The arrows correspond to edges with values above 0.52 of a greyscale network which was calculated using PDC. 133
8.5 A few pictures from the standardized set of 260 pictures of Snodgrass and Vanderwart. 134
8.6 **Left:** Average network across trials of a control subject. **Right:** Average network across trials of an alcoholic subject. All networks were calculated at alpha band using mscoherence. Only edges with values above 0.59 are shown. Higher values are shown with red colors and wider edges. The channels Fpz, AFz, Fz, FCz, Cz, CPz, Pz, POz and Oz are not shown in this figure. . . . 135

8.7 **Left:** Average network across control subjects and trials. **Right:** Average network across alcoholic subjects and trials. All networks were calculated at beta band using the RIM method. Only edges with values above 0.42 are shown. Higher values are shown with red colors and wider edges. In both figures we see that due to network averaging the strongest edges connect geographically neighboring channels. 136
8.8 L (left figure) and F (right figure) networks at mu band for subject a. The networks were calculated using the nonlinear RIM measure. Only edges with values above 0.8 are shown. 139
8.9 Subject a and mu band. **a)** Network L minus I (left figure) [resp. F minus I (right figure)]. The networks were calculated using the RIM measure. The color of a vertex shows the (average) PSD of L (resp. F) state. Red colors correspond to higher values than blue colors. **b)** Network I minus L (left figure) [resp. I minus F (right figure). The networks were calculated using PDC. **c)** Ranking of vertices of network I minus L (resp. I minus F) according to Hubbell's centrality index. 140

List of Tables

8.1 Clustering coefficient statistics. The networks were calculated using the RIM method 6.5.3 at beta frequency band. Clustering coefficient was calculated for each trial of each subject of alcoholic or control group. Then the mean and standard deviations were calculated for each group. The last line of the table has "yes" when there is statistically important differentiation between control and alcoholic clustering coefficient and "no" otherwise. . . . 137

8.2 Assortativity statistics. The statistics were calculated as in table 8.1. 137

8.3 Network efficiency statistics. The statistics were calculated as in table 8.1. Additionally in the calculations we take into account the effect of the non-increasing function used to transform edge strength into edge length. Three functions were considered: the inverse function, the $\log_2$ function and the inverse threshold function that assigns length 1 to an edge and length $+\infty$ to absence of an edge between two vertices. The inverse threshold function is the usual graph theoretic distance function applied to graph constructed using the threshold function $\text{thres}_\theta(x)$, where $\theta = 0.4$. 137

Chapter 1

Introduction

The term *network* is overloaded and each scientific and technological subfield using networks has a different working definition. Communication engineers think of networks as systems of routers and switches; sociologists think of networks as influence diagrams representing the social interactions among humans; marketing business people think of networks as populations of buyers; and the physicist thinks of networks as models of phase transition, magnetism, and so on. Biologists use the network metaphor to understand epidemics, genetics, and metabolic systems within cells, and power engineers think of electrical power grids. Neuroscientists use networks to model the structure and function of neural systems. The Committee on Network Science for Future Army Applications, commissioned by the Board on Army Science and Technology in cooperation with the National Research Council of the National Academies [76], describes a network "by its structure (e.g., nodes and links), and its behavior (what the network 'does' as a result of the interactions among the nodes and links)." This operational definition identifies two key ingredients of networks: 1) the structure of a collection of nodes and links that represent something real, and 2) the dynamic behavior of the aggregation of nodes and links. Nodes might be humans, brain regions, neurons, molecules, genes, routers, transformers (in power grids), Web pages, software modules, cities, airports, or research publications. Links might be friendships, contagions, statistical dependencies, interactions, signals, nervous, synapses, cables, Internet links, roads, flights, or bibliographical citations.

In this thesis we investigate algorithms for the analysis and visualization of the structure of biomedical networks. Network structural information is summarized into a few numbers called network statistics or network measures using efficient algorithms. Network statistics help domain experts to classify networks and relate their structure with their function. Visualization is also an effective tool for the understanding of networks. Good visualization reveals the hidden structure of the networks and amplifies human understanding, thus leading to new insights and new findings.

Networks are prevalent in life sciences. In molecular biology, high-throughput experimental techniques has led to the generation of huge data sets that can be either represented as networks or interpreted in the context of networks. Gene-regulatory and signal-transduction networks [108, 222, 11], protein interaction networks [180, 89], metabolic networks [64, 200], and phylogenetic networks

[113, 45] are a few examples of biological networks. Even the knowledge itself in molecular biology can be conceptualized and described in algorithmically useful ways with the help of ontologies [143]. Ontologies divide a specific knowledge domain into classes or kinds. Complex knowledge domains require multiple levels of hierarchically organized classes and are structured as directed acyclic graphs. Biological domain ontologies are authored and maintained by scientists to categorize the extraordinarily complex vocabulary in molecular biology and to support the retrieval, integration and analysis of their data. Gene Ontology (GO) [19], the most conspicuous example, has been enjoying the status of a de facto standard vocabulary for the annotation of experimental data pertaining to the attributes of gene products. The GO has been widely applied to data drawn from experiments involving organisms and biological processes of many different types. It has also been subject to a series of logical reforms, which have enhanced the degree to which it can it be exploited for algorithmic purposes. The GO is now routinely used in gene expression analyses of a wide range of biological phenomena, including phenomena relevant to our understanding of human health and disease. The GO is the union of three ontologies, each representing a key concept in molecular biology: the *Molecular Function* of gene products; their role in multi-step *Biological Processes*; and their localization to *Cellular Components*. The building blocks of the three ontologies are the terms which consist of a numerical identifier, a name and a number of attributes such as a definition. At the time that this thesis was written the three ontologies contained 8658, 17453 and 2480 terms respectively. The ontologies are structured as a directed acyclic graph (DAG) where the set of vertices is the set of terms and an edge is a relationship from a more specialized term to a less specialized term. GO terms can be linked by five type of relationships: *is_a*, *part_of*, *regulates*, *positively_regulates* or *negatively_regulates*. Due to its huge size, visualizing the whole GO with the usual node-link representation leads to visual clutter. On the other hand space filling visualizations, such as the treemaps, have the capacity to display thousands of items legibly in limited space via a two-dimensional rectangular map. Treemaps have been used to visualize the Gene Ontology by first transforming the DAG into a tree [194]. However this transformation has several undesirable effects such as producing trees with a large number of nodes and scattering the rectangles associated with the duplicates of a node around the display rectangle. In this thesis we study the problem of visualizing a DAG with space filling techniques without converting it to a tree first.

In Chapter 2 we define the constraints for a visualization that extends the treemap techniques [49, 183, 30, 127] to DAGs and where the vertices and edges of a DAG are drawn as rectangles. We use the term *DAGmap* to describe space filling visualization according to the constraints and we show that there are DAGs that admit and DAGs that do not admit DAGmap drawings. Moreover we prove that deciding whether or not a DAG admits a DAGmap drawing is NP-complete. Then we study three special cases of the DAGMAP problem by restricting: a) the input to the class of two terminal series parallel digraphs [208], b) the input to the class of minimal vertex series parallel digraphs [208] and c) the allowed partitions of the rectangles (the rectangles are sliced in one dimension, say the vertical). We show that in the first two special cases the admissibility question can be answered in linear time with respect to the input size and we propose linear time drawing algorithms. As far as the third special case is concerned, which we call ONE-DIMENSIONAL DAGMAP problem, we

prove that when the input DAG is layered planar which has its sources on the top layer and its sinks on the bottom layer then the admissibility question can be answered in linear time. However a complete characterization of the class of DAGs that admit a one-dimensional DAGmap is given in the next chapter.

In Chapter 3 we study the class of DAGs that admit a one-dimensional DAGmap. We prove that a DAG admits a one-dimensional DAGmap if and only if it admits a directed ε-visibility representation. Then we give a characterization of the DAGs that admit directed ε-visibility representations. The are the planar DAGs that admit a planar embedding where the sources and the sinks appear on the boundary of the external face. Due to the one-to-one connection between one-dimensional DAGmaps on the one hand and directed ε-visibility representations on the other hand this characterization is valid for the class of DAGs that admit one-dimensional DAGmaps as well. Then we give linear time testing and drawing algorithms. Finally we show that a DAGmap defines a directed three-dimensional ε-visibility representation of a DAG and that the converse does not hold.

In Chapter 4 we connect the DAGmap admissibility problem with the computation of dominators of a DAG which is a well known problem in compiler design and optimization. We show that in DAGmaps the rectangle of a vertex is included in the rectangle of its immediate dominator and we propose a heuristic algorithm that decomposes an st-graph into component st-graphs. Each component st-graph can be drawn separately and then the drawings are combined. When a component st-graph does not admit a DAGmap then it is converted (via vertex duplications) into a two terminal series parallel digraph that admits a DAGmap. When an st-graph has dominators or post-dominators then our algorithm produces significantly fewer duplications than the transformation into a tree. To make this chapter self-standing we briefly review the literature on the computation of dominators and we describe a very simple algorithm which in practice runs in almost linear time.

In Chapter 5 we describe our software for drawing DAGmaps. The chapter starts with the algorithms that we propose for implementing nesting in trees and in two terminal series parallel digraphs. Then we describe a program called *DAGmap View* which displays DAGmaps for classes of DAGs that admit DAGmaps. DAGmap View implements many novel features such as decomposed layout and hierarchy presentation functions. The decomposition of these two functions allows the precalculation of the layout of all vertices of a hierarchy and the navigation through the hierarchy without changing the position of the rectangles and therefore the mental map of the user. In DAGmap view the hierarchy structure is shown using nesting.

In our study we also consider the analysis and visualization of brain functional networks. Motivated by the fact that a program that analyze and visualize brain functional networks is of interest to researchers working in functional connectivity we created BrainNetVis. In the following, brain functional networks are defined and Chapters 6, 7 and 8 are outlined. In these chapters there are descriptions of how to estimate brain functional networks, of how to model, analyze and visualize them, as well as on the BrainNetVis program.

In neuroscience, nervous systems are complex networks capable of generating and integrating information from multiple external and internal sources in real time. The nervous system of the hermaphrodite nematode *caenorhabditis elegans* is very simple; as a result the detailed structure of its neuronal network

composed of 302 neuron cells and about 14 synaptic couplings per neuron, has been obtained by biologists through direct observations [5, 212, 152]. In comparison to the C. elegans neural network, the complexity of the human brain is gigantic: It contains about 10^{11} neuron cells, each of which is connected to $10^3 - 10^4$ other neurons via synaptic couplings. Construction of the detailed map of all neuron connections in human brain is unrealistic and will be so in the near future [187]. Recording or tracing 10^{15} connections is not only technically impossible, it may also be unnecessary. Indeed there is overwhelming evidence that human cognitive functions depend on the activity and synchronization of large populations of neurons in distributed networks [46]. Anatomically segregated brain regions containing large population of neurons with similar functional involvement and interregional pathways possibly represent the most appropriate organizational level for the brain structural or anatomical network [187]. At this level of analysis brain regions correspond to vertices and interregional pathways correspond to edges. Characterization of the architecture of the anatomical network in the human brain is crucial since this anatomical substrate supports the dynamic emergence of coherent physiological activity that can span the multiple spatially distinct brain regions. Functional interactions between brain regions make up a *functional network*. Functional networks are thought to provide the physiological basis for information processing and mental representations [52] and have been studied across different conditions of rest [93, 71, 209] or cognitive load [27]. Brain functional networks are estimated from multivariate time series of brain activity where each vertex is identified with a scalar time series and there is an edge between two vertices if and only if statistical relationship is inferred between the corresponding time series [75, 96]. There are many challenging issues when estimating functional networks. Network vertices can easily be identified with functional MRI (fMRI) data but the dependence between vertices can be measured only for low frequencies (< 0.2 Hz) due to the limited time resolution of fMRI. On the other hand the time resolution of electroencephalogram (EEG) and magnetoencephalogram (MEG) is excellent but the mapping from generators in the brain to sensors on the scalp is complex and the topology of a network in sensors space is different from the topology in generators space. Therefore identifying vertices with signals at sensors' space is a poor choice [101]. On the other hand to locate generators from EEG/MEG data one has to solve the inverse problem which is ill posed, sensitive to noise and does not have a unique solution.

In this thesis we calculate brain functional networks from EEG at sensors space to avoid the complications of the inverse problem. However our analysis and our methods are not restricted to sensors space and our discussion is valid for the generators space as well [172]. In Chapter 6 we briefly discuss how to estimate functional connectivity from multivariate EEG signals. Our approach is based on the theory of synchronization of weakly coupled dynamical systems. We describe three synchronization measures. The first is a robust interdependence measure which is based on the theory of recurrence of non-linear dynamical systems. The second is coherence and the third is partial directed coherence which is a frequency domain measure of Granger causality. In our study we also investigated other synchronization measures such as the synchronization likelihood [188], the phase locking value [138] and the directed transfer function [105, 106] but due to space limitation we discuss only three measures that are well established and/or robust to noise. The interested reader should

refer to [170] for a more complete discussion on synchronization measures.

After estimating statistical relationship between every pair of vertices a functional network is formed by keeping only those links that are statistically significant. Although each link depends on time, space, cognitive task and many more details, approximating it by the existence or not of a link between the corresponding vertices is still a very informative representation of the entire system. During the last decade it became evident that brain functional networks are characterized by the same topological properties that are present in most real networks, as for instance relatively small characteristic path lengths, high clustering coefficients, fat tailed shapes in the degree distributions, degree correlations, and the presence of motifs and community structures [1, 191, 184].

The description of brain functional networks is enhanced by considering the values of interactions between vertices and not only the existence or not of a link. Motivated by this fact, in Chapter 7 we examine how existing graph measures and algorithms can be extended to networks that have real values associated with their edges. Especially we study networks where the edge values belong to the interval $(0, 1]$ and interpreted as similarity or synchronization between the corresponding nodes. We call these networks *greyscale networks* since their edge values can be visualized with shades of grey. In greyscale networks edge values encode similarity while most graph algorithms work with edge lengths or weights. For this reason we investigate how to convert edge similarity into length and we define the notion of distance between pairs of nodes. Then based on the assumption that central vertices are functionally important we focus our attention in measures that rank vertices according to their role in a network and according to their attributes. Finally we study network level statistics that capture important topological properties of a network. These statistics can be used to characterize a network and to show that two networks differ.

In Chapter 8 we describe BrainNetVis that visualizes brain functional greyscale networks and computes a number of network statistics. Then we apply the results of Chapter 7 on greyscale networks to functional networks derived from two EEG data sets. The first data set arises from a large study to examine EEG correlates of genetic predisposition to alcoholism. By visualizing functional networks of control and alcoholic subjects, derived from EEG data that was recorded during working memory tasks, we see differences between the two groups associated with brain dysfunction in alcoholic subjects. The differentiation between the alcoholic and control subjects' networks is also revealed by network statistics which could be used as biomarkers in a decision support system for clinical validation. The second data set is from Brain Computer Interface (BCI) competition IV. The EEG data was recorded, in a continuous session, while a subject performed motor imagination tasks (without feedback). We found that it is hard to differentiate between two tasks based on the corresponding functional networks at sensors' space. However by comparing these networks with the idle state network, which is used as null model, some differences are revealed. Our work is one of the first works to study functional connectivity in BCI data.

Chapter 2

Space Filling Visualization of Directed Acyclic Graphs

2.1 Motivation: Visualizing the Gene Ontology

The Gene Ontology project (GO) [19], provides a controlled vocabulary to describe gene and gene product attributes in an organism. Due to its huge size, visualizing the whole GO with the usual node-link representation leads to visual clutter (Fig. 2.2). The reason for this clutter is that, the node-link representations do not make optimal use of the available space since most of the pixels are used for the background. On the other hand, space filling techniques make optimal use of the available space and have the capacity to show thousands of items legibly. At the core of a space filling visualization is a layout function that takes as arguments a list of k positive numbers $\{x_1, x_2, \ldots, x_k\}$ and a rectangle R and returns a partition of R into rectangles $R_1, R_2, \ldots, R_k$, where $area(R_i)$ is proportional to x_i. The number of possible partitions is huge and finding a solution such as to minimize the size of the perimeters of the rectangles is NP-complete [29]. Heuristic algorithms produce partitions with desirable properties in reasonable time. For example, the squarified layout, introduced by Bruls et al. [49], strives to produce rectangles with aspect ratio as close as possible to one, where $aspect_ratio = max(width/height,\ height/width)$.

In the case of treemaps the nodes of a tree are visualized as rectangles whose area is proportional to a numeric attribute of the nodes with the property that the value of the attribute in a parent node is equal to the sum of the values of

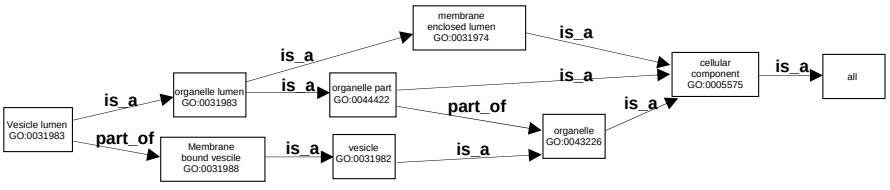


Figure 2.1: A small subgraph of the GO DAG.

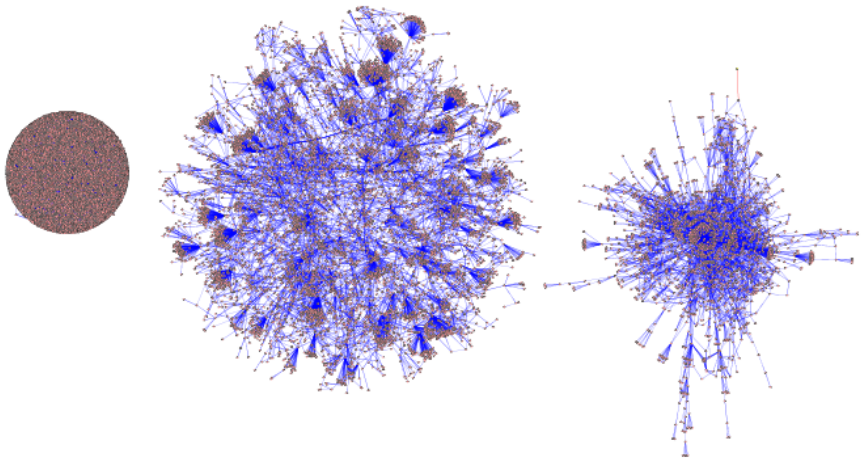


Figure 2.2: Node-link representation of the whole GO. GO consists of three three structured and controlled vocabularies: "biological process", "molecular function" and "cellular component". These vocabularies are shown in this order from left to right.

its children nodes. The rectangle that represents the root of a tree is partitioned into rectangles representing its children and the algorithm is repeated recursively for each node of the tree [30, 49, 183].

In a treemap the hierarchy structure is presented using several approaches including the nested [102, 183], the cascaded [123, 127], and the cushion [213] presentations.

Treemaps have been used to visualize compound graphs that contain both hierarchical (rooted tree) relations and adjacency relations [92]. Space filling techniques are used for the hierarchical or inclusion relations and lines or curves for the adjacency relations.

In the context of GO, treemaps have been used to visualize microarray data, where each gene transcript is assigned all possible paths that start from it and terminate to the most general term (the "all" term) of GO [23]. Symeonidis et al. [194] proposed to decompose the complete GO DAG into a tree by duplicating the vertices with many incoming edges, and then to use a treemap algorithm to visualize the tree (Fig. 2.10). The duplication of a vertex however triggers the duplication of all of its out-neighbors. Therefore the transformation of a DAG into a tree leads to trees with (potentially exponentially) many more nodes than the original DAG. At the time that this thesis was written the initial GO DAG had 28591 vertices, while the produced equivalent tree had 1136854 nodes. Another drawback of duplicating the vertices is that the rectangles associated with the multiple replicas of a vertex are scattered around the display rectangle (Fig. 2.3).

In this thesis we introduce the problem of drawing a DAG using space filling

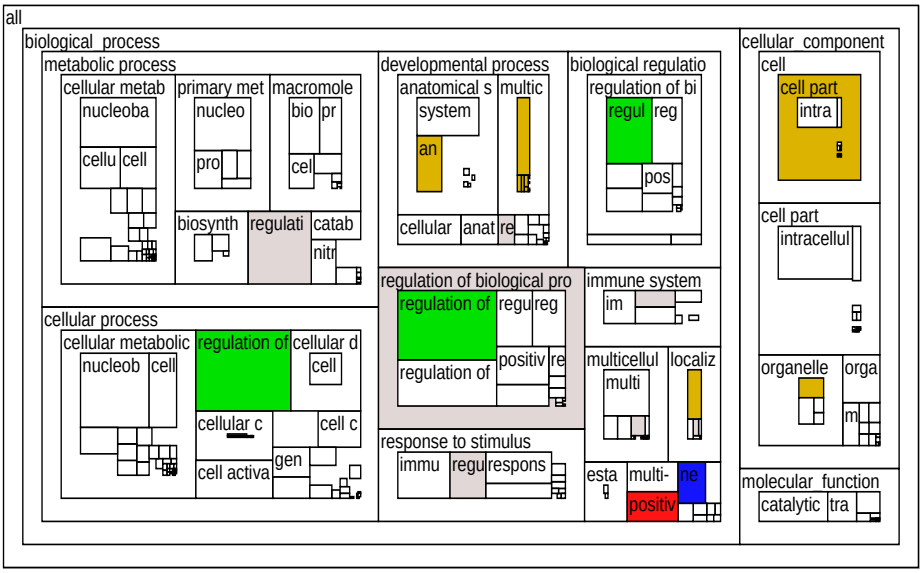


Figure 2.3: Transforming the GO DAG into a tree and then drawing it as a treemap. The first five layers from the root are shown. The tree structure is visualized via nesting. The multiple copies of the node "regulation of cellular process" are shown with green background color. The color of nodes refers to the relationship between a GO term and its parent GO term. We use white color for *is_a* relationship, light brown color for *part_of*, gray color for *regulates*, red color for *positively_regulates* and blue color for *negatively_regulates*. E.g., "metabolic process" *is_a* "biological process" which *is_a* "all". Also "cell part" *is_a* "cellular component" and *part_of* "cell".

techniques without converting it to a tree first. We consider several variations of the problem, we present some characterizations of simple families of DAGs that admit such a drawing, and provide complexity results for the general problem.

2.2 Preliminaries: Elements of Graph Theory

A *graph* $G = (V, E)$ is a pair of *vertices* $V = \{v_1, \ldots, v_n\}$ and *edges* $E = \{e_1, \ldots, e_m\}$, where each edge $e \in E$ is an ordered or unordered pair of vertices. An ordered pair $e = (u, v) \in V \times V$ is called a *directed edge*, while an unordered pair $e = \{u, v\}$, where $u, v \in V$, is called an *undirected edge*. In case $u = v$, e is called a (directed or undirected) *loop*. A graph with no loops is called *loop-free*. If E is a multiset, edges with multiplicity greater than one are called *multiple edges*. If we explicitly allow multiple edges, the graph is called a *multigraph*. A graph without loops and multiple edges is called *simple*. A graph in which each edge is directed, is called a *directed graph* or *digraph*. A digraph with multiple edges is called *directed multigraph* or *multidigraph*.

Given a graph $G = (V, E)$, the cardinality of V is usually denoted by n, the cardinality of E by m. The two vertices joined by an edge are called its *endvertices*. In the undirected case, if two vertices u, v are joined by an edge

$e = \{u, v\}$ then they are *adjacent* and we call them *neighbors*. The vertices u and v are *incident* with edge e. Two edges are *adjacent* if they have exactly one common endvertex. The *degree* of a vertex $v \in V$, denoted by $deg(v)$, is the number of edges in E that have v as an endvertex. The sets of neighbors of vertex v is denoted by $N(v) = \{u \in V \mid \{v, u\} \in E\}$ and the sets of its incident edges is denoted by $\Gamma(v) = \{e \in E \mid v \in e\}$. Note that $deg(v) = |\Gamma(v)|$.
In the directed case, if $e = (u, v) \in E$ is a directed edge, we say that e is *incident from* u (or *outgoing from* u) and *incident to* v (or *incoming to* v); vertex u is the *origin* of e and vertex v is the *destination* of e. The origin of e is denoted by $orig(e)$ and the destination of e by $dest(e)$. The *out-degree* of $v \in V$, denoted by $deg^+(v)$, is the number of edges in E that have origin v. The *in-degree* of $v \in V$, denoted by $deg^-(v)$, is the number of edges with destination v. The *degree* of $v \in V$, denoted by $deg(v)$, is the number of edges with origin or destination v ($deg(v) = deg^+(v) + deg^-(v)$). For every vertex $v \in V$, $N^+(v) = \{u \in V \mid (v, u) \in E\}$ and $N^-(v) = \{u \in V \mid (u, v) \in E\}$ are the sets of *out-neighbors* and *in-neighbors* of vertex v, respectively. Analogously, $\Gamma^+(v) = \{e \in E \mid orig(e) = v\}$ and $\Gamma^-(v) = \{e \in E \mid dest(e) = v\}$ are the sets of *edges incident from* and *to* vertex v, respectively. Note that $deg^+(v) = |\Gamma^+(v)|$ and $deg^-(v) = |\Gamma^-(v)|$.

A *walk* from v_0 to v_k in a graph $G = (V, E)$ is an alternating sequence $v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k$ of vertices and edges, where $e_i = \{v_{i-1}, v_i\}$ in the undirected case and $e_i = (v_{i-1}, v_i)$ in the directed case. The length of the walk is defined as the number of edges on the walk. There is always a zero-length walk from v_0 to v_0. The walk is called a *path*, if $e_i \neq e_j$ for $i \neq j$, and a path is a *simple path* if $v_i \neq v_j$ for $i \neq j$. If there is a path p from v_0 to v_k, we say that v_k is *reachable* from v_0 via p and in the directed case we write $v_0 \overset{p}{\rightsquigarrow} v_k$. A path with at least one edge and $v_k = v_0$ is a *cycle*. A cycle is a *simple cycle* if $v_i \neq v_j$ for $0 \leq i < j \leq k - 1$. A *directed cycle* graph is a cycle in a digraph, with all the edges being oriented in the same direction. A *directed acyclic graph*, also called a *DAG*, is a directed graph with no directed cycles.

A topological numbering of G is an assignment of numbers to the vertices of G, such that for every edge (u, v) of G, the number assigned to v is greater than the one assigned to u (i.e., $number(v) > number(u)$). If the edges of G have nonnegative weights assigned with them, then the number assigned to v is greater than or equal to the number assigned to u plus the weight of (u, v) (i.e., $number(v) \geq number(u) + weight(u, v)$). The numbering is optimal if the range of numbers assigned to vertices is minimized.

A layering of a graph G is a partition of V into subsets $L_1, L_2, \ldots, L_h$, such that if $(u, v) \in E$, where $u \in L_i$ and $v \in L_j$, then $i > j$. A DAG with a layering is a layered DAG. The span of an edge (u, v) with $u \in L_i$ and $v \in L_j$ is $i - j$. The DAG is proper if no edge has a span greater than one. A DAG G can be made proper by replacing each long edge (u, v) of span $k > 1$ with a path $u = v_1, v_2, \ldots, v_k = v$, adding the dummy vertices $v_2, \ldots, v_{k-1}$ [60]. Also, we denote the set of edges incident from the nodes of a layer L_i, $i \in \{2, \ldots, h\}$ by E_i ($E_i = \cup_{u \in L_i} \Gamma^+(u)$).

A *drawing* $\mathcal{G}$ of a graph (digraph) G is a function which maps each vertex v to a distinct point $\mathcal{G}(v)$ and each edge (u, v) to a simple open Jordan curve $\mathcal{G}(u, v)$, with endpoints $\mathcal{G}(u)$ and $\mathcal{G}(v)$. A drawing is *planar* if no two distinct edges intersect. A graph is planar if it admits a planar drawing. A (planar)

embedding is an equivalence class of planar drawings and is described by the circular order of the neighbors of each vertex. An *embedded graph* is a graph with a specified embedding.

2.3 Problem Definition

2.3.1 Drawing Constraints

Treemaps display a tree hierarchy via the inclusion invariant. Namely, the drawing rectangle of any node (different from the root) is included within the drawing rectangle of its parent. When the graph is a DAG, the above invariant should be replaced by the invariant that the drawing rectangle of any vertex is included within the union of the rectangles of its in-neighbors. Apart from this invariant it is plausible to assume that the drawing rectangles of two vertices do not overlap when each node is not reachable from the other and that the drawing rectangle of a vertex is covered by the drawing rectangles of its out-neighbors. Another observation is that in DAGs an edge may be visualized as a rectangle which is contained in the intersection of the origin and the destination vertex rectangles.

Let R_u denote the drawing region of a vertex $u \in V$ and similarly R_e denote the drawing region of an edge $e \in E$. Then the above invariant and assumptions are summarized in Definition 2.1.

Definition 2.1 (DAGmap drawing) *A DAGmap drawing of a DAG $G = (V, E)$ is a space filling visualization of G that satisfies the following drawing constraints:*

- *B1. Every vertex is drawn as a rectangle (R_u is a rectangle for every $u \in V$).*
- *B2. The union of the rectangles of the sources of G form a partition of the initial drawing rectangle ($R = \cup_{s \in S} R_s$ and $\forall s_1, s_2 \in S$ with $s_1 \neq s_2$ $area(R_{s_1} \cap R_{s_2}) = 0$, where $S \subset V$ is the set of sources of G).*
- *B3. Every edge is drawn as a rectangle that has positive area ($\forall e = (u, v) \in E$, R_e is a rectangle and $area(R_e) > 0$).*
- *B4. The rectangle of every non-source vertex $u \in V$ is equal to the union of the rectangles of edges incident to u ($R_u = \cup_{e \in \Gamma^-(u)} R_e$).*
- *B5. The rectangles of edges incident from a non-sink vertex $u \in V$ form a partition of the rectangle of u ($R_u = \cup_{e \in \Gamma^+(u)} R_e$ and $\forall e_1, e_2 \in \Gamma^+(u)$ with $e_1 \neq e_2$ $area(R_{e_1} \cap R_{e_2}) = 0$).*

Theorem 2.1 *In a DAGmap drawing of DAG $G = (V, E)$, if for some pair of edges $e_1, e_2 \in E$ with $e_1 \neq e_2$, it holds that that $orig(e_1)$ is not reachable from $dest(e_2)$, $orig(e_2)$ is not reachable from $dest(e_1)$, then the rectangles R_{e_1} and R_{e_2} do not overlap (i.e., $area(R_{e_1} \cap R_{e_2}) = 0$).*

Proof: Assume for a contradiction that there are two edges $a_1, b_1 \in E$, where $a_1 \neq b_1$, such that $orig(a_1)$ is not reachable from $dest(b_1)$, $orig(b_1)$ is not reachable from $dest(a_1)$ and $area(R_{a_1} \cap R_{b_1}) > 0$. The proof is accomplished by the following recursive procedure which ends after a finite number of steps.

If $orig(a_1) = orig(b_1)$ from constraint B5 we arrive at contradiction. If both vertices $orig(a_1)$ and $orig(b_1)$ are sources we arrive at contradiction from constraint B2. For this reason we assume that $orig(a_1) \neq orig(b_1)$ and at least one of vertices $orig(a_1)$, $orig(b_1)$ is not a source. We have three cases. Case 1: if both $orig(a_1)$, $orig(b_1)$ are non-source vertices, then from constraint B5 we have that $R_{a_1} \subset R_{orig(a_1)}$ and $R_{b_1} \subset R_{orig(b_1)}$. Then $R_{a_1} \cap R_{b_1} \subset R_{orig(a_1)} \cap R_{orig(b_1)}$ $\Rightarrow 0 < area(R_{a_1} \cap R_{b_1}) \leq area(R_{orig(a_1)} \cap R_{orig(b_1)})$. Then from constraint B4 we have that there is an edge a_2 incident to vertex $orig(a_1)$ and an edge b_2 incident to vertex $orig(b_1)$ such that $area(R_{a_2} \cap R_{b_2}) > 0$. We repeat the above procedure for edges a_2 and b_2. Case 2: $orig(a_1)$ is a source vertex and $orig(b_2)$ is a non-source vertex. Then from constraint B4 there exists an edge b_2 incident to vertex $orig(b_1)$ such that $area(R_{a_1} \cup R_{b_2}) > 0$. We repeat the procedure for edges a_1 and b_2. The third case where $orig(a_1)$ is a non-source vertex and $orig(b_1)$ is a source vertex is similar to case 2 and we repeat the procedure for vertices a_2 and b_1.

After a finite number of steps the above procedure ends and we get two edges a_k and b_l such that $area(R_{orig(a_k)} \cap R_{orig(b_l)}) > 0$ and $orig(a_k) = orig(b_l)$ or $orig(a_k)$, $orig(b_l)$ are both source vertices which is a contradiction. □

From constraints B1-B5 and Theorem 2.1 it is trivial to prove that:

Proposition 2.1 *In a DAGmap drawing of a DAG G the following hold:*

a) The rectangle of every non-source vertex $u \in V$ is contained in the union of rectangles of its in-neighbors ($R_u \subset \cup_{v \in N^-(u)} R_v$).

b) The rectangle of every non-sink vertex $u \in V$ is covered by the rectangles of its out-neighbors ($R_u \subset \cup_{v \in N^+(u)} R_v$).

c) Every edge rectangle is contained in the intersection of the origin and destination vertex rectangles ($\forall e = (u, v) \in E$, $R_e \subset R_u \cap R_v$).

d) For every pair of vertices $u, v \in V$ if there is no path from u to v and from v to u then their rectangles R_u, R_v do not overlap ($area(R_u \cap R_v) = 0$).

In light of Theorem 2.1 and Proposition 2.1 it is clear that Definition 2.1 is equivalent to DAGmap definition that we proposed in [207].

The drawing rules of Definition 2.1 are quite general since they do not constrain the area of the sink vertices, or how the area of a vertex is distributed to its incoming edges. To simplify the analysis of the problem we constrain these two parameters by making the following assumptions.

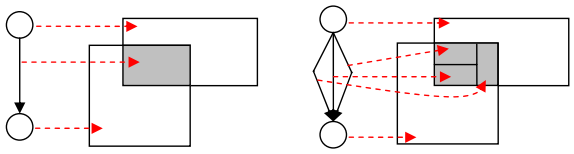
Definition 2.2 (Additional drawing constraints)

A1. The sink vertices are drawn in equal area rectangles.

A2. The rectangles of the edges incident to a vertex have equal areas (For every non-source vertex u and every $e \in \Gamma^-(u)$, $area(R_e) = \frac{area(R_u)}{|\Gamma^-(u)|}$).

In real applications, we may choose to draw only vertex rectangles, only edge rectangles, or both. We usually draw edge rectangles when the DAG has multiple edges or when the edges carry out important information such as the type of relationship between two vertices. See Fig. 2.5 for an example.

Having defined the drawing rules, we can define the following problems:



(a) Edge drawing (b) Drawing of parallel edges

Figure 2.4: An example where an edge rectangle is the intersection of the origin and destination vertex rectangles. In the case of multigraphs, if there are k parallel edges between an origin vertex and a destination vertex, the intersection rectangle is arbitrarily partitioned into k equal area rectangles.

1. Given a DAG G_1, does G_1 admit a DAGmap?
2. In case that the answer to the first problem is negative, what is the minimum number of vertex duplications that are needed to transform G_1 into a DAG G_2 that admits a DAGmap?

2.3.2 Examples and Counter-Examples of DAGs that Admit a DAGmap

Examples of DAGs that admit a DAGmap appear in Fig. 2.6. From the counter-example of Fig. 2.7 we see that there are DAGs that do not admit a DAGmap drawing. The DAG in Fig. 2.7(a) cannot be drawn due to adjacency constraint violation. The first-layer vertices e, f, g, h, i, j constrain the pairs of second-layer vertices $\{a, b\}$,$\{a, c\}$,$\{a, d\}$,$\{b, c\}$,$\{b, d\}$,$\{c, d\}$ to be drawn in adjacent rectangles. However we cannot have such a configuration. In this case in order to draw the

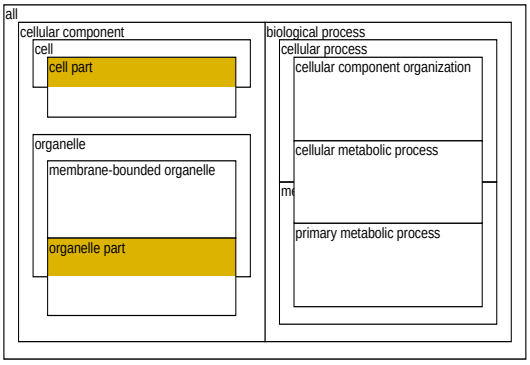


Figure 2.5: In this example a subgraph of the GO DAG is drawn. The color refers to the relationship between two GO terms. We use white color for *is_a* relationship and light brown color for *part_of* relationship. The term "cellular metabolic process" *is_a* "cellular process" and *is_a* "metabolic process". The term "cell part" *is_a* "cellular component" and *part_of* "cell". (Compare the visualization of "cell part" between this figure and Fig. 2.3.)

DAG we can either duplicate one of the vertices e, f, g, h, i, j or relax some of constraints of Definition 2.1, as we did in Fig. 2.7(b) where we relaxed constraint B1. However, allowing the set of possible drawings of a vertex to include any simply connected region of the plane complicates the problem without offering a guarantee that a DAG will admit a DAGmap. See Lemma 2.1 for a counter-example which is based on the four colors (map coloring) theorem. The two-layer DAG of Fig. 2.8, which has five second-layer vertices and ten first-layer vertices (one sink for every pair of sources), does not admit a DAGmap if the vertices are drawn as simply connected regions.

Lemma 2.1 *The DAG of Fig. 2.8 does not admit a DAGmap.*

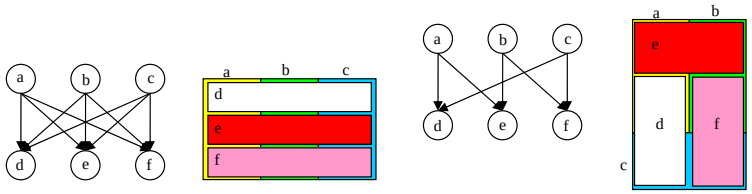
Proof: Suppose that the DAG of Fig. 2.8 admits a DAGmap. Then regions R_a, R_b, R_c, R_d and R_k are pairwise adjacent. However this contradicts with the Four Colors Theorem [163]. □

2.3.3 Vertex Duplication

Usually, a DAG encountered in practice does not admit a DAGmap. In this case we should relax one or more of constraints B1-B5, A1-A2 or change the form of the DAG. Symeonidis et al. [194] chose to transform the DAG into a forest of trees by performing vertex duplications. An example of a vertex duplication is shown in Fig. 2.9, where after the creation of two replicas of vertex h the DAG of Fig. 2.7(a) is transformed into a new DAG which admits a drawing.

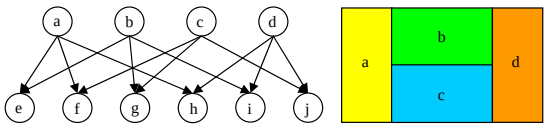
2.3.4 Exponential Increase in the Number of Vertices

Transforming a DAG G into a tree via vertex duplication (see Fig. 2.10) guarantees the admissibility of the drawing but may lead to an exponential increase in the number of vertices. In a worst case scenario, DAG G has n vertices arranged in n layers (Fig. 2.11). For simplicity we use the same numbering for vertices and for layers. DAG G has $n \cdot (n-1)/2$ directed edges. There is a directed edge from every vertex i to every vertex j with $i > j$. Suppose that G is transformed into a tree having T_n nodes. Then the relation between n and T_n is:
$T_1 = 1 = 2^0$, $T_2 = 1 + T_1 = 2^1$, $T_3 = 1 + T_2 + T_1 = 2^2$
And by induction on n:
$T_n = 1 + T_{n-1} + T_{n-2} + \ldots + T_1 = T_{n-1} + T_{n-1} = 2 \cdot T_{n-1} = 2 \cdot 2^{n-2} = 2^{n-1}$

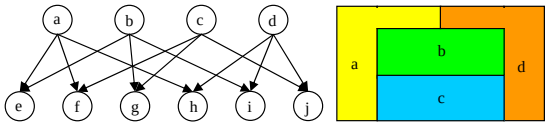


(a) The $K_{3,3}$ DAG (b) A "circular" DAG

Figure 2.6: Examples of DAGs that admit DAGmaps.



(a) Adjacency constraints violation



(b) Relaxing constraint B1

Figure 2.7: Counter-example of a DAG that does not admit a drawing that satisfy all constraints B1-B5. However by relaxing constraint B1 the DAG admits a DAGmap.

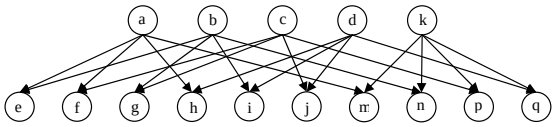


Figure 2.8: Counter-example of a DAG that does not admit a drawing that satisfy constraints B2-B5 if the vertices are drawn as simply connected regions of the plane.

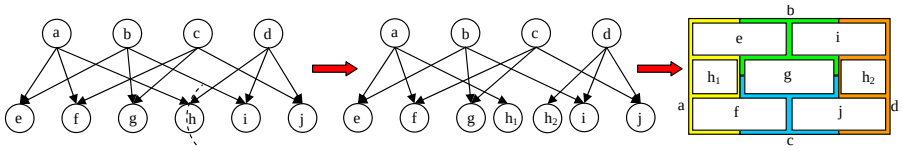


Figure 2.9: After the duplication of vertex h the DAG of Fig. 2.7(a) is transformed into a DAG that admits a DAGmap.

2.4 Treemap Algorithms

To create a treemap, one must define a tiling algorithm, that is, a layout function that divides a rectangle into sub-rectangles of specified areas. To date, many primary rectangular treemap algorithms have been developed including Ordered [30], Slice And Dice [183], Squarified [49], and Strip [30]. To make this thesis self-contained we will briefly describe two of the above algorithms. The Slice and Dice algorithm that was the algorithm proposed in the seminal paper of Shneiderman [183], and the Squarified algorithm [49] that produces legible treemap layouts with low aspect ratios.

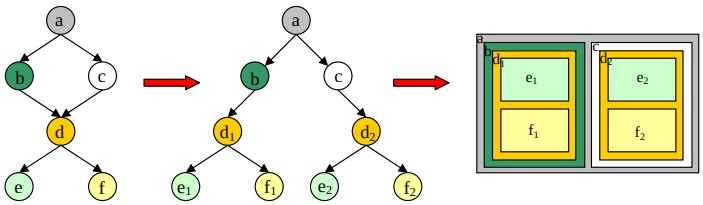


Figure 2.10: Example of transforming a DAG into a tree and then drawing it as a treemap.

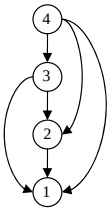


Figure 2.11: A DAG with four vertices, having the structure described in the worst case scenario.

2.4.1 Treemap Generic Algorithm

Treemap algorithms have a generic recursive form and specialize in the layout function. We will first present the generic form (see Algorithm 1). We assume that each tree node v is associated with a numeric attribute called $size$ and a rectangle r. Rectangle r consists of the coordinates x and y of upper left corner and of the rectangle's width w and height h (i.e., $r = (x, y, w, h)$).

2.4.2 Slice and Dice Algorithm

The Slice and Dice algorithm proposed by Shneiderman [183] follows the generic Algorithm 1 and uses a layout function that slices a rectangle in the vertical or the horizontal direction. The slicing direction alternates between consecutive layers and is determined by input parameter $horiz$. A pseudo-code implementation of the layout function appears in Algorithm 2.

2.4.3 Squarified Algorithm

The Squarified algorithm proposed by Bruls et al. [49] follows the generic Algorithm 1 and uses a layout function that produces rectangles with low aspect ratio using a greedy strategy. A pseudo-code implementation of the layout function appears in Algorithm 3. This implementation follows the implementation which is publicly available in the web page on the history of treemaps which is maintained by Ben Shneiderman [1]. An alternative implementation can be found in Appendix II. Algorithm 3 runs in time $O(klog(k))$, where k is the number of

[1] http://www.cs.umd.edu/hcil/treemap-history/

Algorithm 1 Treemap

Input: Tree *root*, and initial drawing rectangle R
Output: Treemap drawing

```
root.r ← R;
treemapRecursive(root, true);

function treemapRecursive(v, axis)
    paintRectangle(v.r);
    if (v.hasChildren()) then
        layout(v.r, v.size, v.children(), axis); // divide rect. v.r into sub-rect.
        axis ← not axis; // change orientation (horizontal/vertical).
        for u ∈ v.children() do
            treemapRecursive(u, axis);
```

Algorithm 2 Slice Layout

Input: Rectangle r, numeric parameter *size*, list of nodes *children*, *horiz*
Output: divide rectangle r into sub-rectangles assigned to nodes in *children*

```
function sliceLayout(r, size, children, horiz)
    b ← 0;
    for v ∈ children do
        a ← v.size/size;
        if (horiz) then // true: cut the horizontal side
            v.r.x ← r.x + r.w * b;
            v.r.w ← r.w * a;
            v.r.y ← r.y;  v.r.h ← r.h;
        else // false: cut the vertical side
            v.r.x ← r.x;  v.r.w ← r.w;
            v.r.y ← r.y + r.h * b;
            v.r.h ← r.h * a;
        b ← b + a;
```

nodes in list *children*. The sorting step, which simplifies the implementation of the algorithm and improves the aspect ratio of the rectangles, can be avoided by assuming that list *children* is sorted according to *size*. Therefore the squarified algorithm can be adjusted to run in time $O(k)$.

2.5 Special Cases

We continue by considering three special cases. The first two cases are based on restricted forms of DAGs, the third case on a restricted way of rectangle partitioning.

2.5.1 Two Terminal Series Parallel Digraphs

A Two Terminal Series Parallel (TTSP) digraph is recursively defined as follows [60, 208].

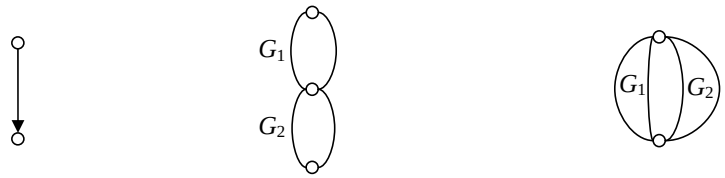
Definition 2.3 (Two Terminal Series Parallel digraphs)

i) A digraph consisting of two vertices joined by a single edge is TTSP (the base graph);

ii) If G_1 and G_2 are TTSP digraphs, so is the digraph obtained by either of the following operations:

a) Series composition: identify the sink of G_1 with the source of G_2.

b) Parallel composition: identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.



(a) The base TTSP digraph

(b) Series composition

(c) Parallel composition

Figure 2.12: Recursive definition of a TTSP digraph.

A TTSP digraph G is naturally associated with a rooted binary tree T, which is called the decomposition tree (or parse tree) of G, and which provides information on how the graph G is constructed using series and parallel compositions.

Definition 2.4 *The decomposition tree T of a TTSP digraph $G = (V, E)$ has three types of nodes: S-nodes, P-nodes and Q-nodes. The leaves of T are Q-nodes and represent base graphs of G. The internal nodes are either P-nodes or S-nodes. T is defined recursively as follows:*

i) If G is a base graph, then T consists of a single Q-node.

Algorithm 3 Squarified Layout

Input: Rectangle r, numeric parameter *size*, list of nodes *children*
Output: divide rectangle r into sub-rectangles assigned to nodes in *children*

sortDescending(*children*); // sort children list according to *size* in descending order
squarifiedLayout($r,\ size,\ children$);

function squarifiedLayout($r,\ size,\ nodes$)
  if ($nodes$.length() < 3) **then** // list *nodes* contain less than 3 nodes
    sliceLayout($r,\ size,\ nodes,\ (r.w \geq r.h)$); // $horiz \leftarrow (r.w \geq r.h)$
    return;

  List $headNodes \leftarrow \emptyset$; // list of nodes; initially empty.
  $bestAspect \leftarrow +\infty;\ b \leftarrow 0;$
  if $(r.w < r.h)$ **then**
    for $i = 1$ **to** $nodes$.length() **do**
      $v \leftarrow nodes$.get(i);
      $a \leftarrow v.size/size;\ b \leftarrow b + a;$
      $aspect \leftarrow \max((b^2 * r.h)/(a * r.w),\ (a * r.w)/(b^2 * r.h));$
      if $(aspect < bestAspect)$ **then**
        $bestAspect \leftarrow aspect;\ headNodes.add(v);$
      else // $aspect \geq bestAspect$
        $b \leftarrow b - a;$
        break;
    $r_1 \leftarrow (r.x,\ r.y,\ r.w,\ b * r.h);$
    $r_2 \leftarrow (r.x,\ r.y + b * r.h,\ r.w,\ (1 - b) * r.h);$
  else // $r.w \geq r.h$
    for $i = 1$ **to** $nodes$.length() **do**
      $v \leftarrow nodes$.get(i);
      $a \leftarrow v.size/size;\ b \leftarrow b + a;$
      $aspect \leftarrow \max((b^2 * r.w)/(a * r.h),\ (a * r.h)/(b^2 * r.w));$
      if $(aspect < bestAspect)$ **then**
        $bestAspect \leftarrow aspect;\ headNodes.add(v);$
      else // $aspect \geq bestAspect$
        $b \leftarrow b - a;$
        break;
    $r_1 \leftarrow (r.x,\ r.y,\ b * r.w,\ r.h);$
    $r_2 \leftarrow (r.x + b * r.w,\ r.y,\ (1 - b) * r.w,\ r.h);$

  sliceLayout($r_1,\ b * size,\ headNodes,\ (r_1.w > r_1.h)$);
  squarifiedLayout($r_2,\ (1 - b) * size,\ nodes$.remove($headNodes$));

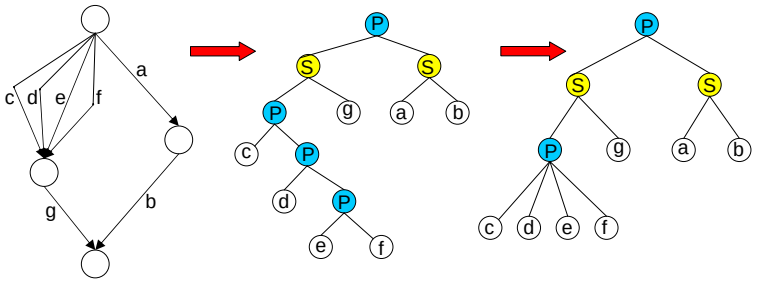


Figure 2.13: Given a TTSP digraph G, we construct its decomposition tree by performing series and parallel reductions. Then we merge the adjacent P-nodes.

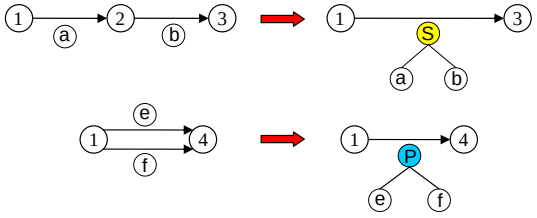


Figure 2.14: Series and parallel reduction rules.

ii) If G is created by a parallel composition of TTSP digraphs G_1 and G_2, let T_1 and T_2 be the decomposition trees of G_1 and G_2 respectively, then the root of T is a P-node and has subtrees T_1 and T_2.

iii) If G is created by a series composition of TTSP digraphs G_1 and G_2, where the sink of G_1 is identified with the source of G_2, let T_1 and T_2 be the decomposition trees of G_1 and G_2, respectively, then the root of T is an S-node and has left subtree T_1 and right subtree T_2.

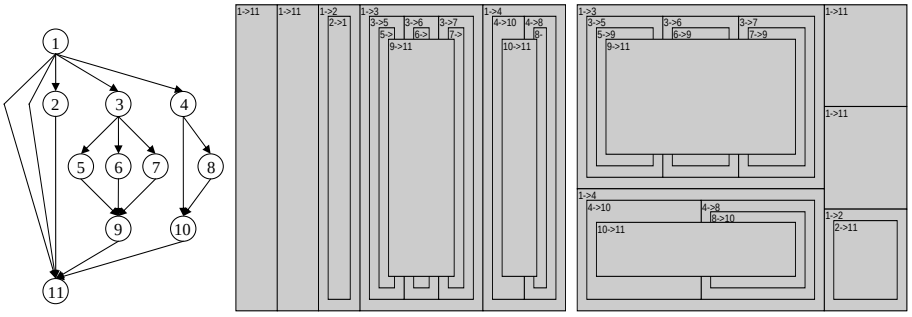
The decomposition tree, which is not unique since several parallel compositions may be combined in different ways and similarly several series compositions, has $O(m)$ nodes and can be computed in $O(m)$ time as a by-product of the series and parallel reduction steps of the TTSP recognition algorithm proposed by Valdes et al. [208]. In order to use the decomposition tree as input to a DAGmap drawing algorithm the neighboring P-nodes are merged to a single node such that the resulting tree may have P-nodes of out-degree larger than two.

The TTSP recognition algorithm [208] maintains a list of vertices that initially includes all vertices except the source and the sink. The algorithm proceeds by removing any vertex v from this list and performing as many parallel reductions on the edges incident to (from) it as it is possible before either leaving the vertex with a single entering edge and a single exiting edge, or discovering that the vertex still has at least two distinct in-neighbors or two distinct out-neighbors. In the first alternative, the vertex is removed by a series reduction and the two vertices adjacent to it added to the unsatisfied list if they are not

there already. This process is repeated until the unsatisfied list becomes empty, at which point the same process is applied to the source and the sink (in order to eliminate any multiple edges between them) before stopping. The unsatisfied list becomes empty, either because all vertices (except source and sink) have been deleted by series reductions or because every remaining vertex has two distinct in-neighbors or two distinct out-neighbors. In the first case the DAG is TTSP; in the second it is not.

In order to obtain the decomposition tree, we associate a label with each edge of the TTSP digraph being reduced. Initially the label of each edge ia a trivial binary tree consisting of a single node. As the reduction process introduces new edges we use rules of Fig. 2.14 to compute the binary trees used to label them.

Algorithm 4 finds a DAGmap drawing of a TTSP digraph. The tricky part



(a) A TTSP

(b) A DAGmap drawing using a layout that slices the rectangles in the vertical direction

(c) A DAGmap drawing using the squarified layout proposed by Bruls et al. [49]

Figure 2.15: Example of a TTSP digraph DAGmap drawing. Only edge rectangles are shown. The hierarchy structure is shown via nesting. For an example where only vertex rectangles are shown see Fig. 2.16(i).

of this algorithm is how to assign sizes to nodes of the decomposition tree. If u is an internal node of the decomposition tree, and $u_1, u_2, \ldots, u_k$ are its children then the constraints are:

1. If u is an S-node then $size(u) = size(u_1) = size(u_2)$.
2. If u is a P-node then $size(u) = size(u_1) + \ldots + size(u_k)$.

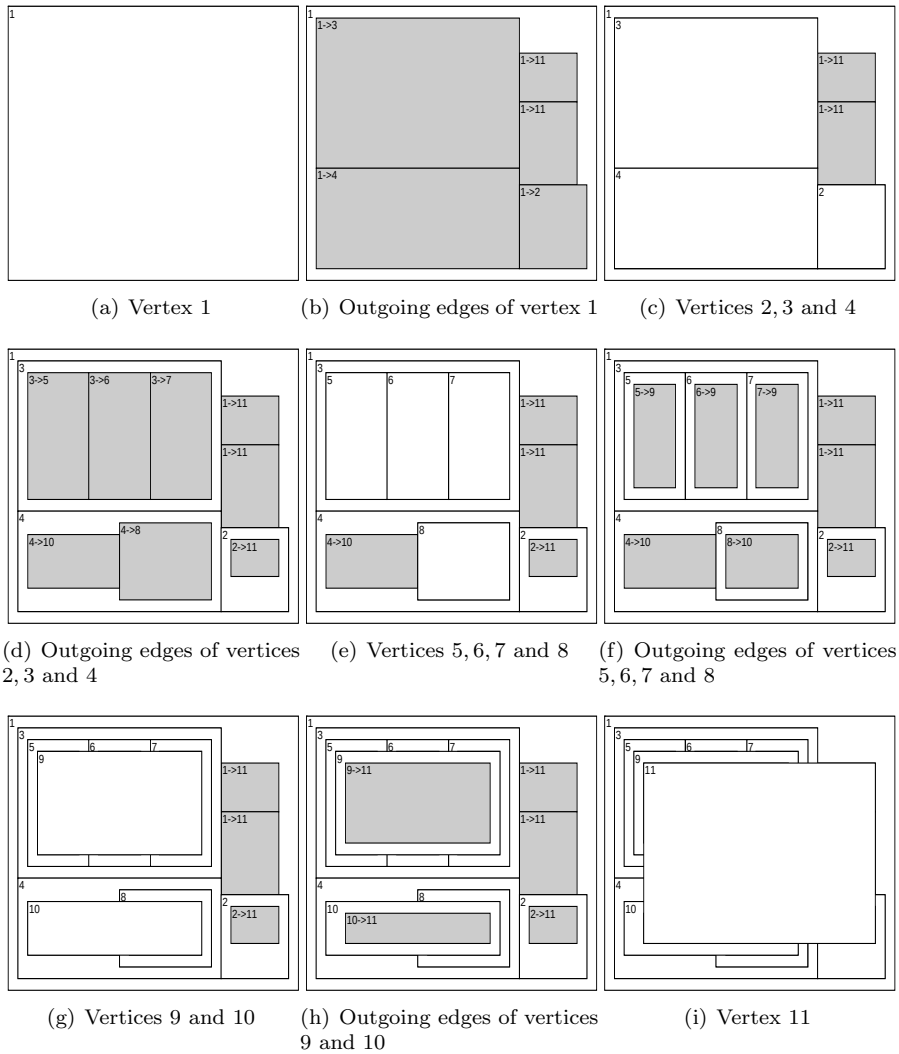
These constraints are not sufficient for a unique solution, and there is some freedom on the choice of the size of some leaf nodes. In the examples of Fig. 2.15 the sizes were calculated using the additional assumption that if among the children nodes of a P-node there are some Q-nodes then all have the same size.

Theorem 2.2 *Every TTSP digraph admits a DAGmap drawing, which can be computed in $\Theta(m)$ time.*

Algorithm 4 TTSP DAGmap drawing

Input: TTSP digraph G and a rectangle R
Output: A DAGmap drawing of G

1. Construct the decomposition tree T of G [208] and merge the neighboring P-nodes. In the resulting tree, P-nodes may have more than two children.
2. S-nodes of T are associated with vertices of G as follows. If an S-node is created as a result of series reduction between two edges (v, u) and (u, w) of G, then associate this S-node with vertex u of G. All vertices of G, apart from the source and the sink, have a corresponding S-node in T.
3. Assign sizes to nodes of the decomposition tree T.
4. Assign rectangle R to the root node of T.
5. Let u be the current node of T and R_u the rectangle assigned to it.
6. If u is an S-node then
   a) Let u_L and u_R be the left and right children of node u.
   b) Assign rectangles $R_L = R_u$ and $R_R = R_u$ to nodes u_L and u_R respectively.
   c) Recursively repeat the procedure from step 5 for nodes u_L and u_R.
7. If u is a P-node then
   a) Let $u_1, u_2, \ldots, u_k$ be the children of u and let $x_1, x_2, \ldots, x_k$ be their sizes.
   b) Call a layout function with input the rectangle R_u and the sizes $x_1, x_2, \ldots, x_k$ to find a partition of R_u into rectangles $R_1, R_2, \ldots, R_k$ where $area(R_i)$ is proportional to x_i.
   c) For $i = 1 : k$
      - assign rectangle R_i to node u_i and recursively repeat the procedure from step 5 for node u_i.
8. When the above recursive procedure finishes, the rectangle assigned to a Q-node of T is also assigned to the associated edge of G. The rectangle assigned to an S-node of T is also assigned to the associated vertex of G.
9. Assign rectangle R to the source and sink of G.
10. Draw vertex and/or edge rectangles according to a priority criterion.



(a) Vertex 1 (b) Outgoing edges of vertex 1 (c) Vertices 2, 3 and 4

(d) Outgoing edges of vertices 2, 3 and 4 (e) Vertices 5, 6, 7 and 8 (f) Outgoing edges of vertices 5, 6, 7 and 8

(g) Vertices 9 and 10 (h) Outgoing edges of vertices 9 and 10 (i) Vertex 11

Figure 2.16: Step by step drawing of the DAG of Fig. 2.15. The vertex rectangles are shown with white color while the edge rectangles are shown with light gray color. The nesting algorithm, used in this example, is slightly different than the nesting algorithm used in Fig. 2.15(c), since care was taken to draw edge rectangles within the intersection of their origin and destination vertex rectangles.

Proof: We will show that the drawing produced by Algorithm 4 is compatible with the constraints of Definition 2.1.

Algorithm 4 assigns a rectangle to every vertex and edge of G. Every edge of G is assigned the rectangle of the associated Q-node of T. The source and the sink of G are assigned the initial drawing rectangle. Now, suppose that u is a vertex of G that is neither a source nor a sink. Then there is at least one edge incident to u and at least one edge incident from u. The TTSP recognition

algorithm after performing a number of series and parallel reductions leaves exactly one edge (v, u) incident to u and exactly one edge (u, w) incident from u. Finally, the TTSP recognition algorithm removes node u by performing a series reduction, in which edges (v, u) and (u, w) are substituted by edge (v, w). The label of edge (v, w) is an S-node having as left subtree the label of edge (v, u) and as right subtree the label of edge (u, w). This S-node of T is associated with vertex u of G and the rectangle assigned to this S-node is also assigned to vertex u.

Clearly, constraints B1 and B2 are satisfied. Constraint B3 is satisfied since Algorithm 4 always divides a rectangle into non-trivial rectangles (For each $e \in E$, $area(R_e) > 0$. If u is the source of G then it is assigned the initial drawing rectangle and therefore $R_e \subset R_u$.

We will show that constraint B4 is satisfied by induction on the composition rules of TTSP digraphs. If G is a base graph composed of two vertices connected by an edge then the rectangle of the sink is equal to the rectangle of the edge and constraint B4 is satisfied. Now, suppose that for two TTSP digraphs G_1 and G_2 constraint B4 is satisfied. Let G be the TTSP digraph that is produced by identifying the sink of G_1 with the source of G_2. The rectangle assigned to the source and the sink of G is equal to the rectangles assigned to the source and sink vertices of G_1 and G_2. Using the induction hypothesis, we conclude that the rectangle of every non-source vertex of G is equal to the union of rectangles of its incoming edges.
Now, let G denotes the TTSP digraph that is produced by parallel composition of G_1 and G_2. Rectangle R_s assigned to source s of G is partitioned into two rectangles R_{s_1} and R_{s_2}. Rectangle R_{s_1} is assigned to source s_1 of G_1 and rectangle R_{s_2} to source s_2 of G_2. Rectangle R_t of the sink t of G is partitioned into two rectangles R_{t_1} and R_{t_2} assigned to sink t_1 of G_1 and to sink t_2 of G_2 respectively. It hold that $R_{t_1} = R_{s_1}$ and $R_{t_2} = R_{s_2}$ because the source and sink rectangles of G_1 (resp. G_2) are equal. Then by the induction hypothesis, constraint B4 holds for every non-source and non-sink vertices of G. It remains to show that constraint B4 holds for sink t of G. The set of edges $\Gamma^-_G(t)$ incident to t of G is equal to the union of the sets of edges $\Gamma^-_{G_1}(t_1)$ and $\Gamma^-_{G_2}(t_2)$ incident to t_1 of G_1 and to t_2 of G_2 respectively. Also $R_t = R_{t_1} \cup R_{t_2}$ and by the induction hypothesis R_{t_1} (resp. R_{t_2}) is equal to the union of the rectangles of edges $\Gamma^-_{G_1}(t_1)$ (resp. $\Gamma^-_{G_2}(t_2)$). Therefore R_t is equal to the union of the rectangles of edges $\Gamma^-_G(t) = \Gamma^-_{G_1}(t_1) \cup \Gamma^-_{G_2}(t_2)$.
The structure of the proof that the constraint B5 is satisfied, is similar to the one for constraint B4. The only difference is that we should also mention that the rectangles of the outgoing edges of a vertex do not overlap and in the parallel case apart from $R_s = R_{s_1} \cup R_{s_2}$ we should also mention that $area(R_{s_1} \cap R_{s_2}) = 0$. Therefore, all constraints of Definition 2.1 are satisfied.

Algorithm 4 runs in time $\Theta(m) + T_{layout}(m)$, where $T_{layout}(m)$ is the total time consumed by the layout function. The TTSP recognition algorithm runs in $\Theta(m)$ time and the drawing algorithm performs one traversal of the decomposition tree which has $\Theta(m)$ nodes. The total time consumed by the layout function is $T_{layout}(m) = \sum_{u \text{ is a P-node}} T_{layout}(deg(u))$. If we assume that $deg(u)$ is bounded then $T_{layout}(deg(u)) = const$ and $T_{layout}(m) = \Theta(m)$. If $deg(u)$ grows as a function of n or m then we can still keep the layout time in $O(m)$ by choosing a linear time layout function such as the slice and dice layout

or the squarified layout 3 without the sorting step. Therefore Algorithm 4 can be computed in time $\Theta(m)$. □

2.5.2 Minimal Vertex Series Parallel Digraphs

Algorithm 4 draws a DAGmap of TTSP digraph G. A natural question is whether this algorithm can be used to draw a DAGmap of a Minimal Vertex Series Parallel (MVSP) digraph which is recursively defined as follows [208].

Definition 2.5 (Minimal Vertex Series Parallel digraphs (MVSP))

i) A digraph having a single node and no edges is MVSP;

ii) If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are MVSP digraphs, so is either of the digraphs constructed by the following operations:

a) Series composition: $G_s = (V_1 \cup V_2, E_1 \cup E_2 \cup (N_1 \times R_2))$, where N_1 is the set of sinks of G_2 and R_2 is the set of sources of G_2.

b) Parallel composition: $G_p = (V_1 \cup V_2, E_1 \cup E_2)$.

Definition 2.6 (Line digraph) *The line digraph of a digraph G is a digraph $L(G)$ that has:*

a) a vertex $f(e)$ for each edge e of G; and

b) an edge $(f(e_1), f(e_2))$ for each pair of edges of G of the form $e_1 = (u, v)$, $e_2 = (v, w)$.

Lemma 2.2 *[208] An acyclic digraph with a single source and a single sink is TTSP if and only if its line digraph is a MVSP dag.*

Definition 2.7 (Complete Bipartite Composite DAGs) *A DAG G is Complete Bipartite Composite (CBC) if there exists a set of complete bipartite subgraphs of G: $B_1, B_2, \ldots, B_k$ called the bipartite components of G, such that:*

a) each edge of G belongs to exactly one bipartite component;

b) every vertex v of G, except the sinks, belongs to the head of exactly one bipartite component denoted $h(v)$;

c) every vertex v of G, except the sources, belongs to the tail of exactly one bipartite component denoted $t(v)$.

Lemma 2.3 *[208] A DAG has a line digraph inverse if and only if it is CBC.*

Lemma 2.4 *[208] Every MVSP DAG is CBC, and therefore has a line digraph inverse.*

Definition 2.8 (The inverse line digraph of a CBC digraph) *Let G be a CBC DAG with bipartite components $B_1, B_2, \ldots, B_k$. The vertex set of $L^{-1}(G)$ is $\{B_\alpha, B_1, B_2, \ldots, B_k, B_\omega\}$ and its edge set has an edge for each node of G defined as follows:*

a) for each source v of G, $L^{-1}(G)$ has an edge $(B_\alpha, h(v))$;

b) for each sink v of G, $L^{-1}(G)$ has an edge $(t(v), B_\omega)$;

c) for each vertex v that is both a source and a sink of G, $L^{-1}(G)$ has an edge (B_a, B_ω); and

d) in all other cases, the edge of $L^{-1}(G)$ that corresponds to vertex v of G is $(t(v), h(v))$.

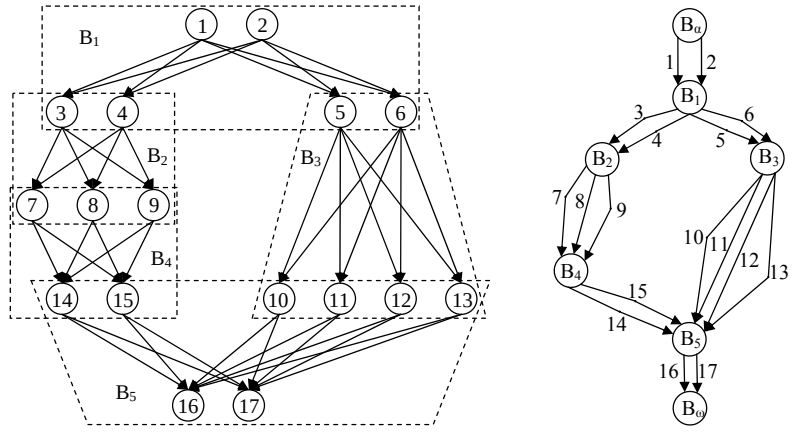
Using the relation between a MVSP digraph and its inverse line digraph we can draw a MVSP digraph using the following algorithm:

Algorithm 5 MVSP space filling drawing

Input: MVSP digraph G
Output: A space filling drawing of G

1. Find the complete bipartite components of G.
2. Find the inverse line digraph $L^{-1}(G)$ of G, using Definition 2.8.
3. Use algorithm 4 to draw the edges of $L^{-1}(G)$. As layout function use the slice layout 2.

The slice layout is the only layout that leads to a DAGmap drawing in some simple cases (see Fig. 2.6(a)). An example of a MVSP digraph and its inverse line digraph are shown in Fig. 2.17. The space filling visualization of this DAG is shown in Fig. 2.18.

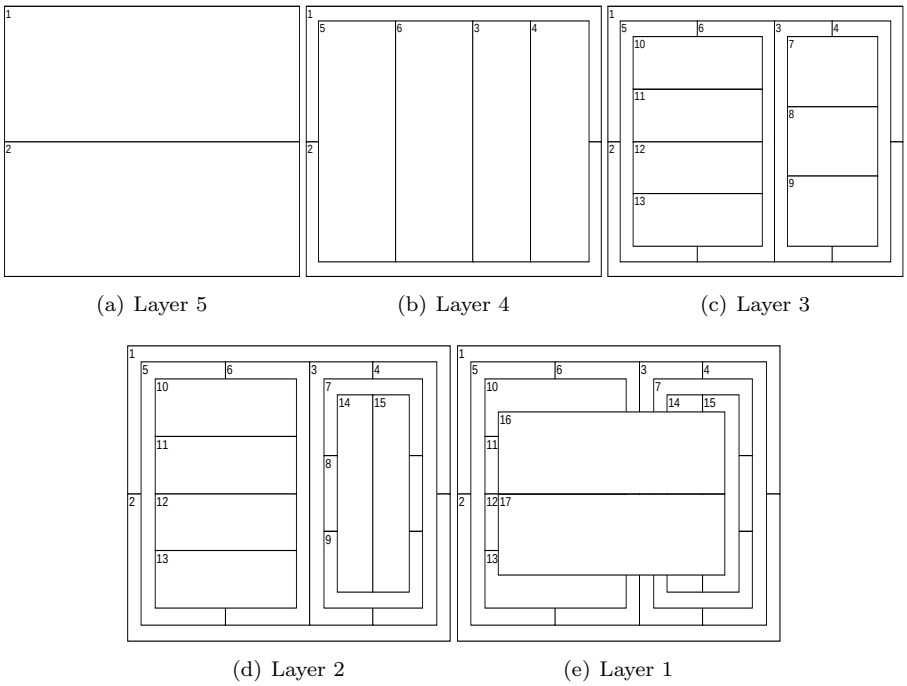


(a) A MVSP digraph and its complete bipartite components

(b) Inverse line digraph

Figure 2.17: Example of finding the inverse line digraph of a MVSP digraph

The drawing produced by Algorithm 5 is not a DAGmap since drawing condition $B3$ is not satisfied in general as can be seen in Fig. 2.18(e). Rectangle R_{16} of vertex 16 has zero area intersection with rectangle R_{12} of vertex 12



(a) Layer 5 (b) Layer 4 (c) Layer 3

(d) Layer 2 (e) Layer 1

Figure 2.18: Space filling visualization of the MVSP digraph of Fig. 2.17

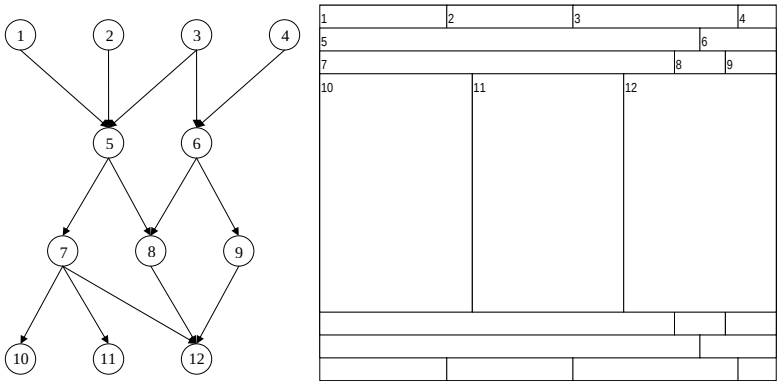
although the DAG has an edge from vertex 12 to vertex 16. Note that there are MVSP digraphs for which Algorithm 5 does produce a DAGmap (For example, consider the DAG of Fig. 2.17(b) and remove vertices 16 and 17 and their incident edges). On the other hand, the inverse line digraph of a MVSP digraph always admits a DAGmap drawing, since it is a TTSP digraph. If we consider the inverse line digraph $L^{-1}(G)$ of G then the drawing of Fig. 2.18(e) is a DAGmap drawing of $L^{-1}(G)$ in which only edge rectangles are shown.

2.5.3 One-Dimensional DAGmaps

We continue by restricting the ways in which the initial rectangle is partitioned; namely we consider only vertical (or only horizontal) partitions.

Definition 2.9 *A DAGmap is called one-dimensional if the rectangles representing the vertices and the edges of a DAG have their top and bottom (left and right) sides on the top and bottom (left and right) sides respectively of the initial drawing rectangle (i.e., the initial rectangle is sliced only in the vertical (horizontal) direction). See Fig. 2.19 and 2.15(b) for examples.*

Since the height (resp. width) of all the rectangles is constant and equal to the height (resp. width) of the initial drawing rectangle, the problem is one-dimensional and the rectangles R_q can be represented by intervals I_q (Fig. 2.20). Next we will define and study the problem of recognizing whether a DAG admits a one-dimensional DAGmap.



(a) A DAG G (b) One-dimensional DAGmap of G

Figure 2.19: An example of a one-dimensional DAGmap drawing of a DAG G. The hierarchy structure is visualized via nesting in the vertical direction.

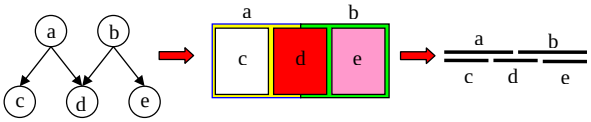


Figure 2.20: A one-dimensional DAGmap example.

Problem 2.1 (ONE-DIMENSIONAL DAGMAP)
INSTANCE: A DAG G.
QUESTION: Does G admit a one-dimensional DAGmap?

A complete characterization of the class of DAGs that admit a one-dimensional DAGmap is given in Chap. 3. In this section we study a restricted version of the ONE-DIMENSIONAL DAGMAP problem. We consider the case that a DAG $G = (V, E)$ is layered with vertex partition $V = L_1 \cup \ldots \cup L_h$, $h > 1$, such that the sources of G are in L_h and the sinks of G are in L_1. Without loss of generality we assume that the layering is proper. To each vertex and edge of G we assign a rational number which is the length of its drawing interval. If $length(I)$ is the length of the initial interval I then each sink vertex $u \in L_1$ is assigned the number $\frac{length(I)}{|L_1|}$, denoted by $size(u)$. The edges incident to a sink vertex u are assigned sizes according to constraint A2 of Definition 2.2 (for each $e \in \Gamma^-(u)$, $size(e) = \frac{size(u)}{|\Gamma^-(u)|}$), the vertices in L_2 are assigned sizes using constraint B5 of Definition 2.1 (for each $u \in L_2$, $size(u) = \sum_{e \in \Gamma^+(u)} size(e)$), and so on. After calculating sizes for vertices and edges the following Lemma holds.

Lemma 2.5 *If the sources of a proper layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$ are in layer L_h, the sinks are in layer L_1 and G has no isolated vertices then:*
a) $\sum_{v \in L_i} size(v) = length(I)$, $i \in \{1, \ldots, h\}$, and
b) $\sum_{e \in E_i} size(e) = length(I)$, $i \in \{2, \ldots, h\}$

Proof: Initially size $\frac{length(I)}{|L_1|}$ is assigned to each sink and the size is propagated toward the sources such that the size of a non-source vertex is equal to the sum of sizes of its incoming edges and the size of a non-sink vertex is equal to the sum of sizes of its outgoing edges. Since there are no sources in layers L_i, $i < h$, no sinks in layers L_i, $i > 1$ and no isolated vertices it is straightforward to see that a) and b) hold. □

To proceed with our analysis we need some definitions. A drawing of a layered graph G in the plane is a layered drawing if the vertices of every L_i, $1 \leq i \leq h$, are placed on a horizontal line $l_i = \{(x, i) \mid x \in \mathbb{R}\}$, and every edge $(u, v) \in E$, $u \in L_i$, $v \in L_j$, $1 \leq j < i \leq h$, is drawn as a line segment between the lines l_i and l_j. A layered drawing of G is called layered planar if no two edges cross except at common endpoints. A layered graph is layered planar if it has a layered planar drawing.

A layered drawing of G determines for every L_i, $1 \leq i \leq h$, a total order $\leq_i$ on the vertices of L_i given by the left to right order of the vertices on l_i. A layered embedding consists of a permutation of the vertices of L_i for every $i \in \{1, \ldots, h\}$ with respect to a layered drawing. A layered embedding with respect to a layered planar drawing is called layered planar.

Theorem 2.3 *Let $G = (V, E)$ be a proper layered DAG with vertex partition $V = L_1 \cup L_2 \cup \ldots \cup L_h$, where $h > 1$, such that the sources are in L_h and the sinks are in L_1. DAG G admits a one-dimensional DAGmap if and only if it is layered planar.*

Proof: Suppose that G admits a one-dimensional DAGmap. Then, by Proposition 2.1, there is a total ordering of the intervals of vertices in L_i, $i \in \{1, \ldots, h\}$. The ordering of the intervals defines an ordering on the vertices of L_i. Therefore the one-dimensional DAGmap defines a layered embedding of G. We will show that this embedding is layered planar. It suffices to show that between two consecutive layers L_i and L_{i-1}, $i \in \{2, \ldots, h\}$, there are no edge crossings. For this, suppose that two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ cross. Then $u_1 <_i u_2$ and $v_1 >_{i-1} v_2$ or $u_1 >_i u_2$ and $v_1 <_{i-1} v_2$. Without loss of generality we assume that $u_1 <_i u_2$ and $v_1 >_{i-1} v_2$. Then $I_{u_1} < I_{u_2}$ and $I_{v_1} > I_{v_2}$. The contradiction comes from the fact that we cannot have $I_{e_1} \subset I_{u_1} \cap I_{v_1}$ and $I_{e_2} \subset I_{u_2} \cap I_{v_2}$, with $length(I_{e_1}) \neq 0$ and $length(I_{e_2}) \neq 0$ and $length(I_{e_1} \cap I_{e_2}) = 0$. We arrived at contradiction because we assumed that two edges in E_i cross. Therefore the embedding is layered planar.

Conversely, suppose that G is layered planar. Then G admits a planar layered embedding. This embedding defines a total order on the vertices of each layer L_i, $i \in \{1, \ldots, h\}$ as well as on the edges between two layers since for $(u_1, v_1), (u_2, v_2) \in E_i$, $i \in \{2, \ldots, h\}$, it holds either $u_1 \leq_i u_2$ and $v_1 \leq_{i-1} v_2$ or $u_1 \geq_i u_2$ and $v_1 \geq_{i-1} v_2$.

Now, suppose that we have calculated the size of vertex intervals and of edge intervals for constructing a one-dimensional DAGmap of G. In a drawing, we arrange the intervals $\{I_v \mid v \in L_i\}$, $i \in \{1, \ldots, h\}$ according to the ordering of vertices in L_i. For $i \in \{1, \ldots, h\}$ the intervals $\{I_v \mid v \in L_i\}$ cover the initial rectangle I and pairwise do not overlap. Therefore they constitute a partition of I. Similarly the ordering of edges in E_i, $i \in \{2, \ldots, h\}$ defines an ordering on the corresponding edge intervals $\{I_e \mid e \in E_i\}$ which form a partition of I. We will show that the size and orderings of vertices and edges of G define a

one-dimensional DAGmap of G by showing that the constraints of Definition 2.1 are satisfied. Clearly constraint B1 is satisfied.
Constraint B2 is satisfied because $\sum_{v \in L_h} size(v) = length(I)$ (see Lemma 2.5).
Constraint B3 is satisfied since every edge $e = (u, v) \in E$ is drawn as an interval I_e of non-zero length.
For constraint B4 we consider how the intervals of vertices in L_{i-1}, $i \in \{2, \ldots, h\}$ are related to intervals of edges in E_i. Let $v_1, v_2, \ldots, v_k$ be the vertices of L_{i-1} arranged in the order defined by the planar layered embedding of G. In the ordering of edges in E_i the incoming edges of v_1 come first. Therefore $I_{v_1} = \cup_{e \in \Gamma^-(v_1)} I_e$. Assuming that for $1 \leq j \leq l < k$ it holds that $I_{v_j} = \cup_{e \in \Gamma^-(v_j)} I_e$ we will show that $I_{v_{l+1}} = \cup_{e \in \Gamma^-(v_{l+1})} I_e$. The union $\cup_{e \in \Gamma^-(v_{l+1})} I_e$ is an interval since the incoming edges of vertex v_{l+1} are consecutive in the ordering of edges E_i. Additionally we have that intervals $I_{v_{l+1}}$ and $\cup_{e \in \Gamma^-(v_{l+1})} I_e$ start at the same point since $I_{v_1} \cup \ldots \cup I_{v_l} = (\cup_{e \in \Gamma^-(v_1)} I_e) \cup \ldots \cup (\cup_{e \in \Gamma^-(v_l)} I_e)$ and they have the same length since $size(I_{v_{l+1}}) = \sum_{e \in \Gamma^-(v_{l+1})} size(I_e) = size(\cup_{e \in \Gamma^-(v_{l+1})} I_e)$. Therefore for every $w \in L_{i-1}$ we have $I_w = \cup_{e \in \Gamma^-(w)} I_e$. Similarly for constraint B5 we can show that for every $w \in L_i$ we have $I_w = \cup_{e \in \Gamma^+(w)} I_e$. Also from Lemma 2.5 it follows that $\forall e_1, e_2 \in \Gamma^+(u)$, $e_1 \neq e_2$ we have that $length(I_{e_1} \cap I_{e_2}) = 0$.
From the above arguments it follows that constraints B4 and B5 are satisfied. □

From the above theorem it follows that the ONE-DIMENSIONAL DAGMAP problem is reduced to LAYER PLANARITY TEST. The later problem can be decided in linear time. When the LAYER PLANARITY TEST algorithm indicates that a layered graph G is layered planar then a planar embedding of G can be obtained in linear time using algorithm LAYER PLANAR EMBED [104].

Algorithm 6 One-Dimensional DAGmap test

Input: a layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$ that is proper and has its sources in layer L_h and its sinks in layer L_1.
Output: "true" if G admits a one-dimensional DAGmap, and "false" otherwise.

1. return LAYER-PLANARITY-TEST(G)

Algorithm 7 One-Dimensional DAGmap draw

Input: planar layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$ that is proper and has its sources in layer L_h and its sinks in layer L_1, and a rectangle R.
Output: DAGmap drawing of G.

1. Find a planar embedding of G using algorithm LAYER-PLANAR-EMBED
2. Assign sizes to vertices and edges of G.
3. Using the orderings and sizes of vertices and edges draw G.

Theorem 2.4 *Every layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$, that is proper and has its sources in layer L_h and its sinks in layer L_1, can be recognized for whether it admits a one-dimensional DAGmap or not in time $O(m)$. If G admits a one-dimensional DAGmap then a drawing of G can be produced in time $O(n)$.*

2.5.4 Minimization of Vertex Duplications in One-Dimensional DAGmaps

Motivated by Algorithms 2 and 3 we pose the question of whether an h-layer graph G_1, that is proper and has its sources in layer L_h and its sinks in layer L_1, can be transformed into a planar h-layer DAG G_2 by performing a minimum number of vertex duplications. We will show that the problem of minimizing the vertex duplications that are needed in order to convert G_1 into G_2 is NP-hard.

Problem 2.2 (DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS)
INSTANCE: A DAG G_1 and an integer K.
QUESTION: Can G_1 be transformed into a DAG G_2 that admits a one-dimensional DAGmap by duplicating at most K vertices?

When the input is restricted to two-layer DAGs where each first-layer vertex has in-degree two, the above problem is related to TWO-LAYER PLANARIZATION problem.

Definition 2.10 *A caterpillar is a connected graph that has a path b called the backbone such that all vertices of degree larger than one lie on b. The edges of a caterpillar that are not on the backbone are the legs of the caterpillar.*

Lemma 2.6 *[67] A two-layer graph $G = (L_1 \cup L_2, E)$ is two-layer planar if and only if it is a collection of disjoint caterpillars.*

Problem 2.3 (TWO-LAYER PLANARIZATION)
INSTANCE: A positive integer K and a two-layer graph $G = (L_1 \cup L_2, E)$.
QUESTION: Can G be made two-layer planar by deleting at most K edges?

Theorem 2.5 *[67] The TWO-LAYER PLANARIZATION problem is NP-complete and remains NP-complete when each vertex in L_1 has degree two.*

Theorem 2.6 *The DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS problem is NP-complete and remains NP-complete even when the input is restricted to simple two-layer DAGs where each first-layer vertex has in-degree two.*

Proof: The problem belongs to NP since given K vertex duplications we can check in linear time if the transformed DAG admits a one-dimensional DAGmap. We will show that the problem is NP-hard by reducing the TWO-LAYER PLANARIZATION problem to it. The reduction is trivial. Let G be a two-layer DAG, where each first-layer vertex has in-degree two. If $e = (u, v)$ is an edge of G then deletion of the edge corresponds to duplication of vertex v. Conversely, duplication of a vertex v corresponds to deletion of one of two edges that are incident to v.

Suppose that the TWO-LAYER PLANARIZATION problem has a solution. Then there are K edges $\{e_1, e_2, \ldots, e_K\}$ whose deletion leads to a graph

G_2 that is two-layer planar. According to Lemma 2.6, the graph G_2 is a collection of disjoint caterpillars. Let $\{v_1, v_2, \ldots, v_K\}$ be the first layer vertices incident to edges $\{e_1, e_2, \ldots, e_K\}$. Now suppose that if instead of deleting the edges $\{e_1, e_2, \ldots, e_K\}$, we duplicate the vertices $\{v_1, v_2, \ldots, v_K\}$, and let $\{v'_1, v'_2, \ldots, v'_K\}$ be the replicas of vertices $\{v_1, v_2, \ldots, v_K\}$ which are attached to edges $\{e_1, e_2, \ldots, e_K\}$. The new graph, call it G_3, is a collection of disjoint caterpillars. This is because if we ignore the edges $\{e_1, e_2, \ldots, e_K\}$ and the incident vertices $\{v'_1, v'_2, \ldots, v'_K\}$, graph G_3 is equal to graph G_2 which is a collection of disjoint caterpillars. Then, since vertices $\{v'_1, v'_2, \ldots, v'_K\}$ have degree one the incident edges $\{e_1, e_2, \ldots, e_K\}$ can be considered as legs attached to the backbone of some caterpillar. This implies that G_3 is a caterpillar and from Lemma 2.6 it follows that G_3 is two-layer planar. Then, from Theorem 2.3, it follows that G_3 admits a one-dimensional DAGmap.

Conversely, suppose that the DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS problem has a solution and let $\{v_1, v_2, \ldots, v_K\}$ be a set of K vertices whose duplication leads to a DAG G_3 that admits a DAGmap drawing. According to Theorem 2.3, DAG G_3 is two-layer planar. DAG G_3 remains two-layer planar if we delete one of the two replicas of vertices $\{v_1, v_2, \ldots, v_K\}$ together with the incident edge. This corresponds to the deletion of K edges from the initial graph G. Therefore the TWO-LAYER PLANARIZATION problem has a solution.

We showed that the DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS problem has a solution if and only if the TWO-LAYER PLANARIZATION problem has a solution. Therefore the problem is NP-complete. □

2.6 The General Case

2.6.1 The Recognition Problem

Suppose that we have a layered DAG. Taking the vertices of a layer L_k isolated from the rest of the DAG, the problem is similar to a floorplan problem where the initial rectangle is dissected into $n_k = |L_k|$ soft rectangles, i.e., rectangles whose area is fixed but their dimensions may vary. The number of possible dissections (the solution space) is bounded below by $\Omega(n_k! 2^{3n}/n_k^4)$ and above by $O(n_k! 2^{5n}/n_k^{4.5})$ [182].

Considering two consecutive layers L_{k+1} and L_k of a DAG, the layouts of the two layers are constrained by the edges among the two layers, according to the drawing rules. The combined solution space may be empty or contain a number of solutions. We will show that deciding whether the solution space is empty or not is NP-complete. We call this decision problem DAGMAP and we define it as:

Problem 2.4 (DAGMAP)
INSTANCE: A DAG G.
QUESTION: Does G admit a DAGmap?

Our hardness result for DAGMAP is based on a transformation from the following decision problem.

Problem 2.5 (3-PARTITION)
INSTANCE: A multiset A of $3m$ *positive integers* $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{3m}\}$ *where*

the α_i's are bounded above by a polynomial in m and $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$, where $\Sigma = \frac{1}{m}(\alpha_1 + \alpha_2 + \ldots + \alpha_{3m})$.
QUESTION: Can A be partitioned into m triples $A_1, A_2, ..., A_m$ such that each triple has the same sum? Specifically each triple must sum to Σ.

3-PARTITION is strongly NP-complete since it remains NP-complete even when representing the numbers in the input instance in unary [78]. The condition $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$ forces every set of α_i's summing to Σ to have size exactly 3.

Theorem 2.7 *The DAGMAP problem is NP-complete even if we restrict it to forests of two-layer DAGs.*

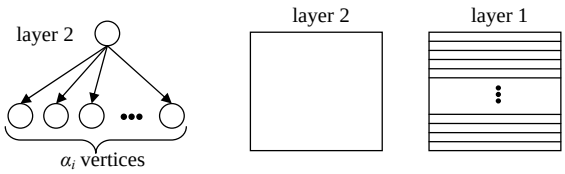
Proof: Given a dissection of the initial drawing rectangle into $|L_k|$ rectangles for each layer L_k, $k \in \{1, \ldots, h\}$ of a DAG G, we can check in polynomial time if these dissections correspond to a DAGmap drawing of G. Moreover, in the VLSI layout literature, there are a few techniques for succinctly encoding the partition of a rectangle into soft rectangles [182]. Therefore the DAGMAP problem belongs to NP. Next we will show that DAGMAP is NP-hard. Given an instance $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{3m}\}$ of 3-PARTITION we will construct a forest of two-layer DAGs that admits a DAGmap drawing if and only if the 3-PARTITION problem has a solution.

Without loss of generality we assume that the first-layer vertices of a DAG are drawn in unit area rectangles. For each $\alpha_i \in A$ we consider a two-layer DAG G_{α_i} which has one vertex in layer two and α_i vertices in layer one. The second-layer vertex is drawn as a rectangle, R_{α_i}, with area α_i, but without any constraint on the aspect ratio of its sides. The first-layer vertices are drawn as unit area rectangles by slicing with parallel horizontal line segments the rectangle of the second-layer vertex (Fig. 2.21). The total area occupied by rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ is $\alpha_1 + \alpha_2 + \ldots + \alpha_{3m} = m\Sigma$.

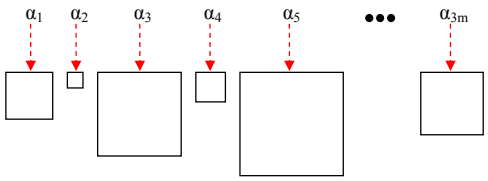
We want to draw rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ on m equal and pairwise isolated rectangular regions inside the initial rectangle R. To do this we consider an additional DAG whose drawing leaves m empty rectangular regions (gaps), each of area Σ. We call this additional DAG the enforcer, since its drawing enforces the previously defined $3m$ rectangles to be drawn inside the m gaps. The shape of the enforcer is unique up to left, right, up or down orientation inside the initial rectangle R (Fig. 2.22). In the following, without loss of generality, we assume that the shape of the enforcer is similar to the one of Fig. 2.22(a).

Enforcer: The DAG used as enforcer (Fig. 2.23) has $2m + 2$ vertices on the second layer, the vertices β and $1, 2, \ldots, 2m + 1$. Every odd numbered vertex has Σ exclusive out-neighbors, while for the even numbered vertices there are no exclusive out-neighbors. Therefore vertices $1, 3, \ldots, 2m+1$ have Σ more area than vertices $2, 4, \ldots, 2m$. The role of vertex β and of first-layer vertices with in-degree greater than one is to align rectangles $R_1, R_2, \ldots, R_{2m+1}$ and to force rectangles R_i and R_{i+1}, $i \in \{1, 2, \ldots, 2m\}$ to be adjacent.

Each of the first-layer vertices $1', 2', \ldots, (2m+1)'$ has two in-neighbors. One is vertex β and the other is the corresponding numbered second-layer vertex. In the drawing rectangle R_β is adjacent to all rectangles $R_1, R_2, \ldots, R_{2m+1}$. Therefore rectangles $R_1, R_2, \ldots, R_{2m+1}$ should be drawn around the sides of rectangle R_β. Rectangle R_γ is drawn on top of rectangles $R_1, R_2, \ldots, R_{2m+1}$

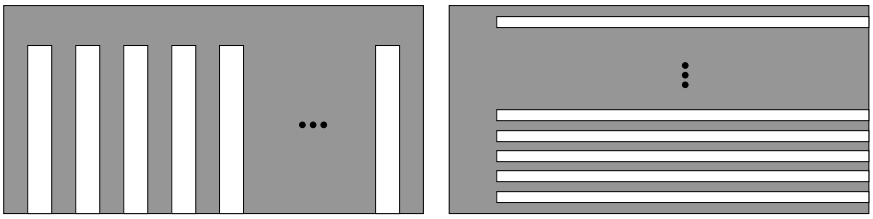


(a) DAG G_{α_i} and one possible drawing of it



(b) An example of the mapping from integers α_i to rectangles of area α_i

Figure 2.21: To each $\alpha_i \in A$ we correspond a DAG G_{α_i} with $\alpha_i + 1$ vertices and to G_{α_i} we correspond a drawing rectangle R_{α_i} of area α_i.



(a) One possible drawing

(b) Another possible drawing

Figure 2.22: The drawing of the enforcer leaves m gaps, each of area Σ.

and has zero area intersection with rectangle R_β. Therefore it forces rectangles $R_1, R_2, \ldots, R_{2m+1}$ to be drawn consecutively along one side of rectangle R_β. Additionally, since the area of rectangle R_γ is equally distributed among rectangles $R_1, R_2, \ldots, R_{2m+1}$, it forces these rectangles to have the same width. The second-layer vertices i and $i+1$, $i \in \{1, 2, \ldots, 2m\}$ have a common out-neighbor which constrains their rectangles to be adjacent. The second common out-neighbor of vertices i and $i+1$, is used for completing the drawing. Also for completing the drawing, vertex 1 has an exclusive out-neighbor and similarly vertex $2m+1$. To sum up, the first layer has $(m+1)\Sigma+6m+4$ vertices and therefore the total area occupied by the drawing of the enforcer is $(m+1)\Sigma+6m+4$ (Fig. 2.24). The total area occupied by the drawings of the enforcer and of DAGs G_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ is: $((m+1)\Sigma + 6m + 4) + m\Sigma = (2m+1)(\Sigma+3) + 1$.

Now suppose that the 3-PARTITION instance has a solution. Then the elements of A can be partitioned into m triples $A_1, A_2, \ldots, A_m$ such that each triple has sum Σ. Then the rectangles that correspond to the elements of a

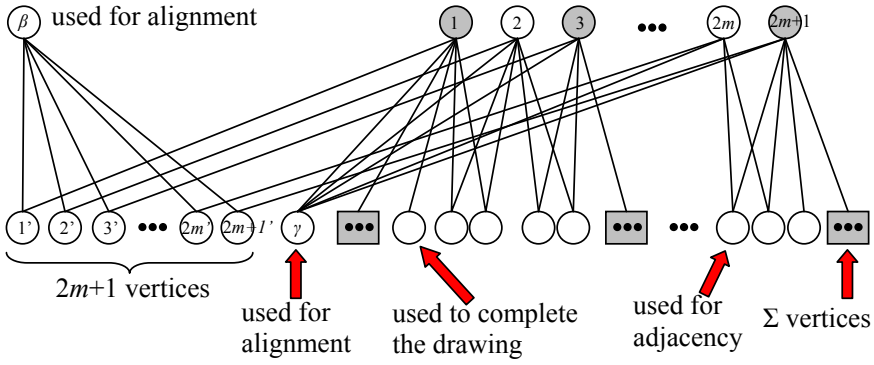


Figure 2.23: The DAG used as enforcer in the proof.

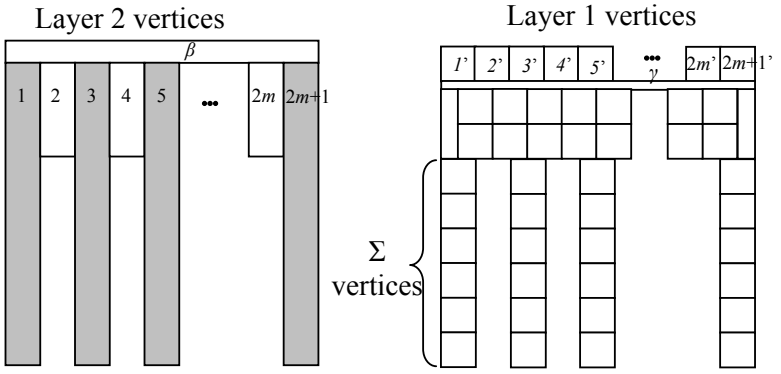


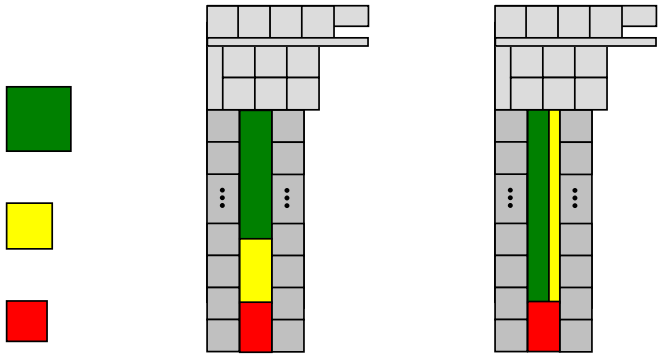
Figure 2.24: One possible drawing of the enforcer.

triple fit exactly into a gap of area Σ (Fig. 2.25). Therefore the DAGMAP problem has a solution.

Conversely, if the DAGMAP problem has a solution then the $3m$ rectangles fill all the gaps. From the condition $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$ on the 3-PARTITION numbers and therefore on the rectangle areas, a gap is filled by exactly three rectangles. Therefore the $3m$ rectangles are partitioned into m triples each of total area Σ. This partition is also a solution to the 3-PARTITION instance. The reduction from the 3-PARTITION to DAGMAP uses a polynomial number of resources since the numbers involved in 3-PARTITION are positive integers bounded by a polynomial in m. Note that one can achieve the geometric construction using simple geometric operations. □

Theorem 2.8 *The DAGMAP recognition problem remains NP-complete even if the input is a two-layer DAG.*

Proof: The reduction is similar to the one in Theorem 2.7. The differences are:



(a) Three rectangles (b) One way to fill the gap (c) Another way to fill the gap

Figure 2.25: An example of how three rectangles, of total area Σ, fill a gap.

1. The enforcer and the $3m$ DAGs of Theorem 2.7 now form a single two-layer DAG instead of a forest of DAGs.
2. The α_i vertices have $\alpha_i - 1$ exclusive out-neighbors instead of α_i.
3. Each of vertices $1, 3, \ldots, 2m+1$ has $\Sigma - 1$ exclusive out-neighbors instead of Σ.

Vertices $1, 3, \ldots, 2m+1$ have Σ more area than vertices $2, 4, \ldots, 2m$. In a drawing, rectangle R_δ is adjacent to each one of rectangles $R_1, R_3, \ldots, R_{2m+1}$, since for $i \in \{1, 3, \ldots, 2m+1\}$, vertices δ and i have one common out-neighbor (Fig. 2.26). Their second common out-neighbor exists for completing the drawing. Rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ have area equal to α_i and are adjacent to rectangle R_δ, since each vertex α_i and vertex δ have a common out-neighbor. Their second common out-neighbor is used for completing the drawing. Similarly, vertex δ has another $(2m+1)(\Sigma-2)+1$ exclusive out-neighbors. Therefore the area of rectangle R_δ is $(2m+1)(\Sigma-2)+1+3m+(m+1) = (2m+1)\Sigma$ and its horizontal and vertical sides have lengths $2m+1$ and Σ respectively.

The adjacency relations between the second-layer vertices $1, 2, 3, \ldots, 2m+1$, β and δ leads to a shape having m gaps, each one of area Σ (Fig. 2.26). The drawing position of rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ is not fixed. The only constraint is that they should be adjacent to rectangle R_δ. However, in order to have a rectangular drawing, triples of rectangles R_{α_i} should fill the gaps. Since rectangles R_{α_i} are adjacent to rectangle R_δ the only way that each of them is drawn inside a gap is with vertical side of length Σ and horizontal side of length $\frac{\alpha_i}{\Sigma}$. The first-layer vertices that are adjacent to vertex δ and to vertex α_i have a drawing with vertical side of length 2Σ and horizontal side of length $\frac{1}{2\Sigma}$.

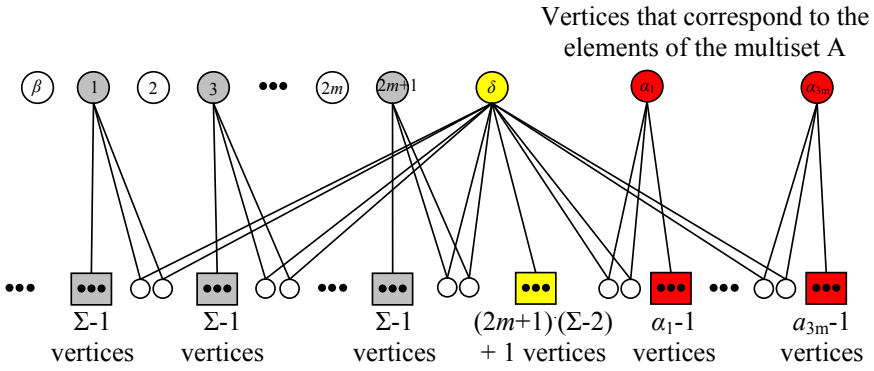
If the 3-PARTITION problem has a solution then triples of rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ fit exactly into the gaps of the drawing and therefore the DAG admits a DAGmap. Conversely, if the DAG admits a DAGmap then rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ fill all the gaps of the drawing. And since

exactly three rectangles can fill a gap, the solution of the DAGMAP instance provides a partitioning of integers α_i into m triples, each of size Σ. □

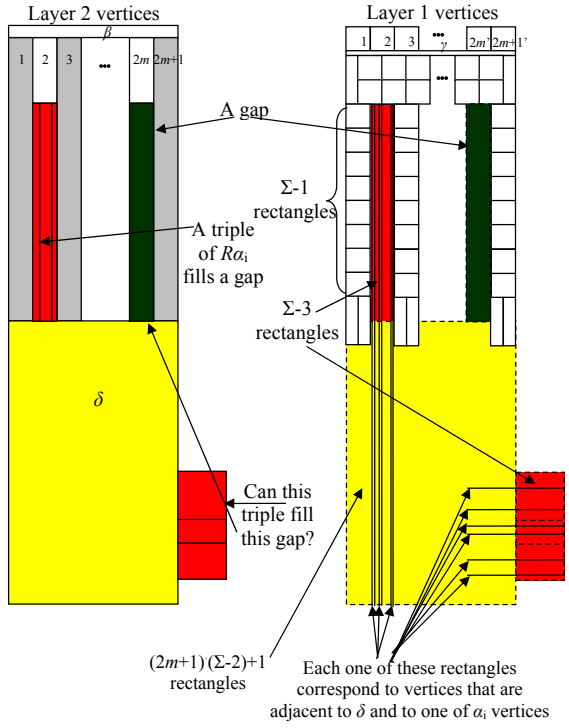
2.6.2 Minimization of Vertex Duplication

Problem 2.6 (DUPLICATIONS IN DAGMAPS)
INSTANCE: A DAG G_1 and an integer K.
QUESTION: Can G_1 be transformed into a DAG G_2 that admits a DAGmap by duplicating at most K vertices.

Comment: The problem DUPLICATIONS IN DAGMAPS is NP-hard since its restriction for $K = 0$ is the problem DAGMAP, which is NP-complete.



(a) Only the new connections are shown



(b) One possible drawing

Figure 2.26: The enforcer of the reduction in Theorem 2.8.

Chapter 3

DAGmaps and Directed ε-Visibility Representations

A visibility representation of a graph G maps vertices of G to sets in Euclidean space and the edges are expressed as visibility relations between these sets. In a two-dimensional visibility representation of a graph the vertices are usually represented by a collection $\mathcal{O}$ of non-overlapping convex regions on the plane and the edges correspond to visibility among pairs of regions. Two points P and Q of the plane are visible if the straight line segment PQ is not obstructed by any region $A \in \mathcal{O}$. Two regions $A, B \in \mathcal{O}$ are called visible if there exist points $P \in A$ and $Q \in B$ which are visible. The problem is simplified if the direction of sight is fixed [116]. In this case there exist complete characterizations of the graphs that admit a visibility representation [199, 61, 215, 60]. Recently, interest has developed in investigating visibility representations in three-dimensions. Relevant to our work is the three dimensional visibility where vertices are represented by disjoint axis-aligned closed rectangles lying in planes parallel to the xy-plane and edges correspond to z-parallel visibility among these rectangles [42]. If graph G is directed then for every edge (u, v) the rectangle of v is below the rectangle of u. Note that in order to be consistent with the downward representation of DAGs we draw the visibility representation downwards whereas in the literature it is drawn upwards [61, 107].

In a DAGmap as well as in a three dimensional rectangle visibility representation of a DAG the vertices are represented by closed rectangles and the edges are closed sets which intersect with the source and destination vertex rectangles. We will show that a DAGmap (resp. a treemap) determines a directed three-dimensional visibility representations of a DAG (resp. tree). Additionally we show that there is a one-to-one correspondence between a one-dimensional DAGmap and a directed ε-visibility representation of a DAG. Using this correspondence we show that the class of DAGs that admit a one-dimensional DAGmap is the class of downward planar digraphs that admit an embedding such that all source and sink vertices appear on the boundary of the external face.

3.1 Preliminaries

Let $\mathcal{S}$ be a set of horizontal non-overlapping segments in the plane. Two segments σ, σ' of $\mathcal{S}$ are said to be visible if they can be joined by a vertical segment not intersecting any other segment of $\mathcal{S}$. Furthermore, σ and σ' are called ε-visible if they can be joined by a vertical band of nonzero width that does not intersect any other segment of $\mathcal{S}$.

Definition 3.1 *A directed (weak) w-visibility representation for a DAG G consists of mapping each vertex v of G into a horizontal segment $\sigma(v)$ (called vertex-segment), and each edge $(u, v) \in E$ into a vertical segment $\sigma(u, v)$ (called edge-segment), so that, the vertex-segments do not overlap, and for each edge $(u, v) \in E$ the corresponding edge-segment $\sigma(u, v)$ has its top endpoint on $\sigma(u)$, its bottom endpoint on $\sigma(v)$, and it does not cross any other vertex-segment $\sigma(a), a \neq u, v$.*

Definition 3.2 *A directed ε-visibility representation for a DAG G is a directed w-visibility representation with the additional property that two vertex-segments are directed ε-visible if and only if the vertex that corresponds to the bottom vertex-segment is adjacent to the vertex that corresponds to the top vertex-segment.*

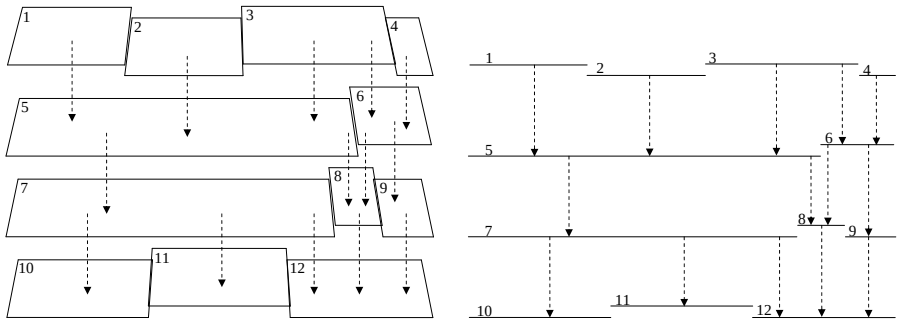
Now consider an arrangement of closed, non-overlapping rectangles in $\mathbb{R}^3$ such that the planes determined by the rectangles are perpendicular to the z-axis, and the sides of the rectangles are parallel to the x-or y-axes. Two rectangles R_i and R_j are ε-visible if and only if between the two rectangles there is a closed cylinder C of positive height and radius such that the ends of C are contained in R_i and R_j, the axis of C is parallel to the z-axis, and the intersection of C with any other rectangle in the arrangement is empty [42].

Definition 3.3 *[165] A directed three-dimensional ε-visibility representation for a DAG G consists of mapping each vertex v of G into a rectangle R_v (called vertex-rectangle), and each edge $(u, v) \in E$ into a vertical closed cylinder C of positive length and radius (called edge-cylinder), so that, the vertex-rectangles do not overlap, and for each edge $(u, v) \in E$ the corresponding edge-cylinder C has its top base on R_u, its bottom base on R_v, and does not intersect any other vertex-rectangle $R_a, a \neq u, v$. Additionally, two vertex-rectangles are ε-visible if and only if the vertex that corresponds to the bottom vertex-rectangle is adjacent to the vertex that corresponds to the top vertex-rectangle.*

An st-graph is an acyclic digraph with a single source s and a single sink t. A planar st-graph is an st-graph that is planar and embedded with vertices s and t on the boundary of the external face.

3.2 One-Dimensional DAGmaps and Directed ε-Visibility Representations

We will show that a DAG admits a one-dimensional DAGmap 2.9 if and only if it admits a directed ε-visibility representation. We recall here that since the height of all the rectangles is constant and equal to the height of the initial



(a) three-dimensional (b) two-dimensional

Figure 3.1: Directed ε-visibility representation of the DAG of Fig. 2.19(a)

drawing rectangle, the combinatorial properties of the problem are unaffected if instead of the vertex and edge rectangles R_q we consider their projections on the horizontal axis. These projections are intervals I_q.

From the vertex rectangles R_q (resp. intervals I_q) of a one-dimensional DAGmap we can construct a directed three-dimensional (resp. two-dimensional) ε-visibility representation by assigning to rectangles (resp. intervals) a z-coordinate. The construction is described in Theorem 3.1. A directed three-dimensional ε-visibility representation of the DAG of Fig. 2.19(a) is shown in Fig. 3.1(a). The corresponding directed (two-dimensional) ε-visibility representation of the DAG is shown in Fig. 3.1(b). The segments of Fig. 3.1(b) are the projections of the rectangles of Fig. 3.1(a) onto the xz-plane.

Theorem 3.1 *If a DAG $G = (V, E)$ admits a one-dimensional DAGmap then it admits a directed ε-visibility representation.*

Proof: Suppose that G admits a one-dimensional DAGmap. From the one-dimensional DAGmap we will construct a directed ε-visibility representation as follows: We compute an optimal topological numbering Y of G, such that only integer numbers are used and the sources are assigned the number 0. We also compute the longest path length, h, in the DAG. Each interval I_u, $u \in V$ of the one-dimensional DAGmap is shifted along the vertical direction and is drawn on the horizontal line with equation $y = y(u) = h - Y(u) + \epsilon \cdot j(u)$, where ϵ is a small positive number (e.g. $0 < \epsilon < \frac{1}{1000 \cdot |V|}$) and $j = j(u) \in \{0, 1, \cdots, |V| - 1\}$ is a unique vertex index. The shifted intervals I_u, $u \in V$ become the vertex-segments $\sigma(u)$ of the ε-visibility representation. The vertex-segments do not overlap since they are drawn on different horizontal lines. Next we add the edge-segments. For each $e = (u, v) \in E$, an edge-segment $\sigma(u, v) = \{(\mu_{uv}, y) \mid y(u) \geq y \geq y(v)\}$ is created, where μ_{uv} is the horizontal coordinate of the middle of the interval I_e. This construction is always possible since $length(I_e) > 0$.

Note that the above construction can also be done using st-numbering instead of an optimal topological numbering. The advantage in this case is that the term $\epsilon \cdot j(u)$ is not needed since the vertex rectangles have distinct y coordinates and therefore are disjoint. The disadvantages are: a) Figures 3.1, 3.2

and 3.4 require more space and b) some proofs become slightly longer since in order to apply Proposition 2.1 we need further checks to show that there is no path between two vertices in the case when two vertices have the same optimal topological numbering.

We will show that edge-segment $\sigma(u, v)$ does not intersect any other vertex-segment apart from vertex-segments $\sigma(u)$ and $\sigma(v)$ at its end points. If $Y(v) - Y(u) = 1$ then there is no vertex segment that crosses with edge-segment $\sigma(u, v)$ due to Proposition 2.1. Therefore suppose that $Y(v) - Y(u) > 1$ and that edge-segment $\sigma(u, v)$ intersects with vertex-segment $\sigma(a)$. Then it follows that a) $Y(u) < Y(a) < Y(v)$ and b) the intervals I_e, $e = (u, v)$ and I_a overlap. Interval I_a is equal to the union of the intervals of edges incident to a. Among those intervals there is one edge interval I_{e_1} that overlaps with edge interval I_e. However there is no path from $dest(e) = v$ to $orig(e_1)$ or from $dest(e_1) = a$ to $orig(e) = u$ since $Y(orig(e_1)) < Y(dest(e))$ and $Y(orig(e)) < Y(dest(e_1))$. This contradicts with Theorem 2.1. Therefore the conditions of w-visibility are satisfied.

Now, suppose that two vertex-segments $\sigma(u)$ and $\sigma(v)$, such that $Y(u) < Y(v)$, are ε-visible while $(u, v) \notin E$. Since $(u, v) \notin E$, it follows that either there is no path from u to v or that there is a path of length at least two. From the ε-visibility of $\sigma(u)$ and $\sigma(v)$ we have that they can be joined by a vertical band $I_\varepsilon \times I_y$ of nonzero width that does not intersect any other segment. Let $e_1, e_2, \ldots, e_k$ be the edges incident from u and $e'_1, e'_2, \ldots, e'_l$ be the edges incident to v. From constraint B5 we have: $I_u = I_{e_1} \cup \ldots \cup I_{e_k}$. Similarly, from constraint B4 we have: $I_v = I_{e'_1} \cup \ldots \cup I_{e'_l}$. Interval I_ε is contained in I_u. Therefore there is an interval $I_e \in \{I_{e_1}, \ldots, I_{e_k}\}$ such that $length(I_e \cap I_\varepsilon) \neq 0$. Since $I_e \cap I_\varepsilon \subset I_v$ it holds that there is at least one interval $I_{e'}$ among $\{I_{e'_1}, \ldots, I_{e'_l}\}$ such that $length(I_e \cap I_\varepsilon \cap I_{e'}) \neq 0$. Then from Theorem 2.1 it follows that there is at least one path from $dest(e)$ to $orig(e')$. It is $dest(e) = a \neq v$ and $I_e \subset I_a$ $\Rightarrow length(I_\varepsilon \cap I_a) \neq 0 \Rightarrow$ the vertical band $I_\varepsilon \times I_y$ intersects with $\sigma(a)$ which contradicts with the hypothesis that $I_\varepsilon \times I_y$ does not intersect any other segment apart from $\sigma(u)$ and $\sigma(v)$.
We proved that if two vertex segments $\sigma(u)$ and $\sigma(v)$, such that $Y(u) < Y(v)$, are ε-visible then $(u, v) \in E$. Therefore the conditions of ε-visibility are satisfied. □

3.3 Characterization of Directed ε-Visibility Representations

Theorem 3.1 reveals an interesting relationship between one-dimensional DAGmaps and ε-visibility representations. Namely a one-dimensional DAGmap defines an ε-visibility representation of a DAG. The converse of this theorem is even more interesting because a) the problem of visibility representation of a DAG has been thoroughly studied and b) it allows us to characterize the class of DAGs that admit a one-dimensional DAGmap. Before we give the converse of Theorem 3.1 we will characterize the class of DAGs that admit a directed ε-visibility representation. This minor result seems to be lacking from the literature. We should mention here that a complete characterization of the class of (undirected) graphs that admit an ε-visibility representation was given by Tamassia and Tol-

lis [199]. A complete characterization of the class of digraphs that admit a (weak) w-visibility representation was given by Di Battista and Tamassia [61].

Theorem 3.2 *[61] A digraph G admits a directed w-visibility representation if and only if G is a subgraph of a planar st-graph.*

For directed ε-visibility representation the following lemma holds.

Lemma 3.1 *If a DAG G admits a directed ε-visibility representation, then there exists a planar embedding $\widehat{G}$ of G such that all source and sink vertices appear on the boundary of the external face.*

Proof: Let $\mathcal{G}$ be a directed ε-visibility representation for G. Construct $\widehat{G}$ by shrinking every vertex-segment of $\mathcal{G}$ into a point, and bending the edge-segments in order to maintain the adjacencies. Suppose, for a contradiction, that there is a source s that does not appear on the boundary of the external face. Then there exists an internal face in which s is not the highest point of its boundary. Then in $\mathcal{G}$ the vertex-segment $\sigma(s)$ is ε-visible from at least one other vertex-segment. Therefore s is not a source. The proof for sink vertices is similar. □

Let C_w and C_ε be the classes of DAGs that admit a directed w-visibility and a directed ε-visibility representation respectively. From the definition of directed ε-visibility representation we have that $C_\varepsilon \subset C_w$. From Lemma 3.1 and Theorem 3.2 it follows that C_ε is properly included in C_w.

Lemma 3.2 *[199] For every vertex v of a planar st-graph G, the incoming (outgoing) edges appear consecutively around v.*

Lemma 3.3 *[199] For every face f of G, the boundary of f consists of two directed paths with common origin and destination.*

Now we will present an algorithm, which is based on the Algorithm Tessellation [60], that computes a directed ε-visibility representation of a planar st-graph. To describe the algorithm we need to introduce some definitions. Let G be a planar st-graph and F be its set of faces (recall that G is embedded). We conventionally assume that F contains two representatives for the external face: the *left external face* s^*, which is incident with the edges on the left boundary of G, and the *right external face* t^*, which is incident with the edges on the right boundary of G. For each element o of $V \cup E \cup F$ we define $orig(o)$, $dest(o)$, $left(o)$, and $right(o)$ as follow:

1) If $o = v \in V$, we define $orig(v) = dest(v) = v$. Also, with reference to Lemma 3.2 we denote by $left(v)$ (resp. $right(v)$) the face that separate the incoming from the outgoing edges of a vertex $v \neq s, t$ in the clockwise direction (resp. counter-clockwise direction). For $v = s$ or $v = t$, we conventionally define $left(v) = s^*$ and $right(v) = t^*$.

2) If $o = e \in E$, we denote by $left(e)$ (resp. $right(e)$) the face on the left (resp. right) side of e. Also, $orig(e)$ (resp. $dest(e)$) denotes the origin (resp. destination) vertex of e.

3) If $o = f \in F$ and $f \neq s^*, t^*$, we denote by $orig(f)$ (resp. $dest(f)$) the vertex that is the common origin (resp. destination) of the two paths forming the boundary of f (see Lemma 3.3). For $f = s^*$ or $f = t^*$, we define $orig(f) = s$ and $dest(f) = t$. Also, we define $left(f) = right(f) = f$.

We define a digraph G^*, associated with planar st-graph G, as follows:

- The vertex set of G^* is the set of faces of G.
- For every edge $e \neq (s,t)$ of G, G^* has an edge $e^* = (f,g)$ where $f = left(e)$ and $g = right(e)$.

Lemma 3.4 *[198] For any two objects $o_1, o_2 \in V \cup E \cup F$ of a planar st-graph G, exactly one of the following holds:*

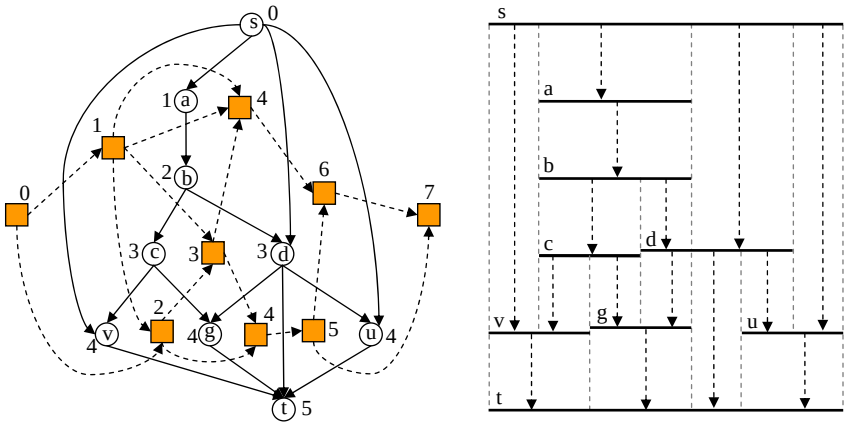
- *G has a directed path from $dest(o_1)$ to $orig(o_2)$*
- *G has a directed path from $dest(o_2)$ to $orig(o_1)$*
- *G^* has a directed path from $right(o_1)$ to $left(o_2)$*
- *G^* has a directed path from $right(o_2)$ to $left(o_1)$*

Algorithm 8 Directed ε-visibility representation

Input: a planar st-graph $G = (V, E)$

Output: a) a directed ε-visibility representation $\mathcal{G}$ of G
b) visibility bands of maximal width that their internal points do not intersect with any vertex-segment of $\mathcal{G}$ and that each one having its top and bottom sides on two vertex-segments $s(u)$ and $s(v)$ respectively if and only if $(u,v) \in E$.

1. Construct the planar st-graph G^*.
2. Compute an optimal topological numbering Y of G such that only integer numbers are used.
3. Compute an optimal topological numbering X of G^*.
4. Let ϵ be a very small positive number e.g. $0 < \epsilon < \frac{1}{1000 \cdot |V|}$
5. $j = 0$;
6. For each vertex $u \in V$, let the coordinates of segment $\sigma(u)$ be:
   $x_L(u) = X(left(u))$; $x_R(u) = X(right(u))$;
   $y(u) = Y(t) - Y(u) + \epsilon \cdot j$; //perturb slightly by adding $\epsilon \cdot j$
   $j = j + 1$;
7. For each edge $e \in E$, let the coordinates of the corresponding maximal visibility band $b(e)$ be:
   $x_L(e) = X(left(e))$; $x_R(e) = X(right(e))$;
   $y_T(e) = y(orig(e))$; $y_B(e) = y(dest(e))$.



(a) Planar st-graphs G and G^*

(b) ε-visibility representation of G and maximal visibility bands

Figure 3.2: Example of a run of Algorithm 8. Planar st-graphs G and G^* are labeled by topological numbering Y and X, respectively.

Theorem 3.3 *Let G be a planar st-graph with n vertices. Algorithm 8 constructs a directed ε-visibility representation in $O(n)$ time.*

Proof: For any pair of vertices $u, v \in V$, the vertex segments $\sigma(u)$ and $\sigma(v)$ do not overlap since they have distinct y coordinates.

For each edge $e = (u, v) \in E$, the corresponding maximal visibility band $b(e)$ has its top side on $\sigma(u)$ since $y_T(e) = y(u)$ and $x_L(u) \leq x_L(e) < x_R(e) \leq x_R(u)$, and its bottom side on $\sigma(v)$ since $y_B(e) = y(v)$ and $x_L(v) \leq x_L(e) < x_R(e) \leq x_R(v)$. We will show that we can choose a vertical band $b'(e) \subset b(e)$ of non-zero width that has its top side on $\sigma(u)$ its bottom side on $\sigma(v)$ and does not intersect with any other vertex-segment $\sigma(a)$. For the coordinates of $b'(e)$ we choose $y'_B = y_B(e) = y(v)$, $y'_T = y_T(e) = y(u)$, and $x_L(e) < x'_L < x'_R < x_R(e)$. If there is a directed path in G from vertex a to vertex $orig(e)$ or a directed path from $dest(e)$ to a then $y(a) > y(u)$ or $y(v) > y(a)$. Therefore vertex segment $\sigma(a)$ and vertical band $b'(e)$ do not intersect. For this we assume that (see Lemma 3.4) G^* either has a directed path from $right(e)$ to $left(a)$ or has a directed path from $right(a)$ to $left(e)$. Then either $x'_R < x_R(e) \leq x_L(a)$ or $x_R(a) \leq x_L(e) < x'_L$ In both cases $b'(e)$ does not intersect with $\sigma(a)$.

Finally since the topological numbering X is optimal the vertex-segment of a non-sink vertex u is covered by the bottom sides of the maximal visibility bands of edges incident from u. Similarly the vertex-segment of a non-source vertex v is covered by the top sides of the maximal visibility bands of edges incident to v. Therefore vertex-segment $\sigma(v)$ is ε-visible from vertex-segment $\sigma(u)$ only if G has an edge (u, v).

The $O(n)$ time bound follows easily since each step of the algorithm can be accomplished in $O(n)$ time. □

Let $G = (V, E)$ be a DAG and $G' = (V', E')$ be the DAG that is formed by augmenting DAG G with two vertices s' and t' and edges from s' to all sources of G and edges from all sinks of G to t' (i.e. $V' = V \cup \{s', t'\}$ and $E' = E \cup \{(s', s) \mid s \text{ is a source of } G\} \cup \{(t, t') \mid t \text{ is a sink of } G\}$). Note that in order to test for st-planarity and find an appropriate planar embedding of G' we add edge (s', t'). This edge constrains the embedding such that vertices s' and t' appear on the boundary of the same face, say the external face. The following theorem holds.

Theorem 3.4 *DAG G admits a directed ε-visibility representation if and only if DAG G' is a planar st-graph.*

Proof: Suppose that G admits a directed ε-visibility representation. Then from Lemma 3.1 there exists a planar embedding $\widehat{G}$ of G such that all source and sink vertices appear on the boundary of the external face. This embedding we can easily be extended to a planar embedding of G'. Therefore G' is a planar st-graph.

Conversely, suppose that G' is a planar st-graph. Using Algorithm 8, we construct a directed ε-visibility representation $\mathcal{G}'$ for G'. If we remove the vertex-segments $\sigma(s')$ and $\sigma(t')$ from $\mathcal{G}'$ we get a directed ε-visibility representation for G. □

Corollary 3.1 *A DAG G admits a directed ε-visibility representation if and only if there exists a planar embedding $\widehat{G}$ of G such that all source and sink vertices appear on the boundary of the external face.*

3.4 Characterization of One-Dimensional DAGmaps

We are now in a position to prove the converse of Theorem 3.1.

Theorem 3.5 *If a DAG $G = (V, E)$ admits a directed ε-visibility representation then it admits a one-dimensional DAGmap.*

Proof: Suppose that G admits a directed ε-visibility representation. Then we compute a directed ε-visibility representation $\mathcal{G}'$ of G' using Algorithm 8 and from $\mathcal{G}'$ a directed ε-visibility representation $\mathcal{G}$ of G by deleting segments $\sigma(s')$ and $\sigma(t')$. From the arrangement of the vertex-segments of $\mathcal{G}$ we will construct a one-dimensional DAGmap drawing of G. To each vertex-segment $\sigma(u), u \in V$ we correspond an interval I_u by taking its projection on the horizontal axis. For each edge $e = (u, v) \in E$ there is exactly one vertical band b of maximal width that has its bottom side on $\sigma(u)$ its top side on $\sigma(v)$ and does not ε-intersect any other segment. The coordinates of b are calculated by Algorithm 8. The edge rectangle I_e is equal to the projection of b on the horizontal axis.

We show that intervals $\{I_u \mid u \in V\}$ and $\{I_e \mid e \in E\}$ satisfy the DAGmap drawing constraints. Drawing constraints B1, B3 are clearly satisfied. Constraint B2 is satisfied due to the optimality of the topological numbering X of G^*. Constraints B4 and B5 are satisfied when the ε-visibility representation of G is produced by Algorithm 8 due to the optimality of the topological numbering X of G^*. □

Combining Theorems 3.1 and 3.5 we have the following theorem:

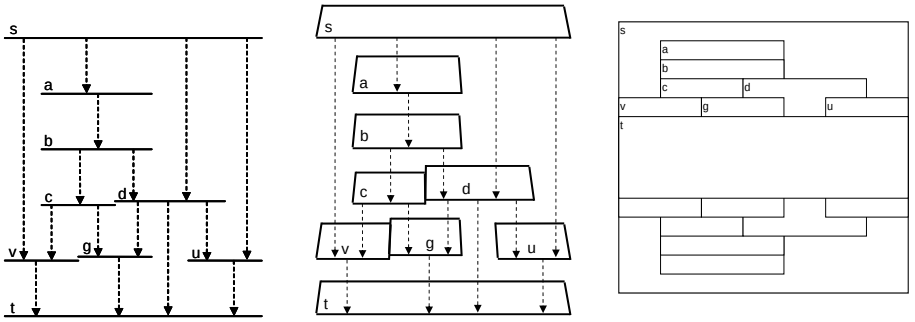
Theorem 3.6 *A DAG $G = (V, E)$ admits a one-dimensional DAGmap if and only if it admits a directed ε-visibility representation.*

Corollary 3.2 *The class of DAGs that admit a one-dimensional DAGmap are the planar st-graphs that admit a planar embedding such that all source and sink vertices appear on the boundary of the external face.*

Algorithm 9 One-dimensional DAGmap drawing

Input: DAG $G = (V, E)$ and drawing rectangle $R = (x_{left}, y_{bottom}, x_{right}, y_{top})$
Output: one-dimensional DAGmap drawing of G if G is admits such a drawing or error message otherwise.

1. From DAG $G = (V, E)$ we construct an st-digraph $G' = (V', E')$, where $V' = V \cup \{s', t'\}$ and $E' = E \cup \{(s', u) \mid u \text{ is a source of } G\ \} \cup \{(u, t') \mid u \text{ is a sink of } G\ \} \cup (s', t')$.
2. If G' is not planar return "G does not admit a one-dimensional DAGmap"
3. Else find a planar embedding of G' such that s' and t' appear on the boundary of the external face.
4. Remove edge (s', t') from G'
5. Call Algorithm 8 with input G' to compute the horizontal coordinates of vertex-segments and maximal visibility bands.
6. Use these coordinates to fill the coordinates of the rectangles of vertex and edge of G



(a) ε-visibility representation of G

(b) Constructing rectangles with sides aligned to xy-axes and constant width

(c) All rectangles are projected onto xy-plane and then nesting is used to display the hierarchy structure

Figure 3.3: Steps 4 and 5 of Algorithm 9 when applied to DAG of Fig. 3.2(a)

Algorithm 9 recognizes whether or not a DAG admits a one-dimensional DAGmap and in the first case it constructs a one-dimensional DAGmap drawing. All steps

of this algorithm can be computed in $O(n)$ time. Therefore we have the following theorem.

Theorem 3.7 *Algorithm 9 computes a one-dimensional DAGmap of a DAG or returns an error message in time $O(n)$ time.*

Proof: For any pair of vertices $u, v \in V$, the vertex segments $\sigma(u)$ and $\sigma(v)$ do not overlap since they have distinct y coordinates.

For each edge $e = (u, v) \in E$, the corresponding maximal visibility band $b(e)$ has its top side on $\sigma(u)$ since $y_T(e) = y(u)$ and $x_L(u) \leq x_L(e) < x_R(e) \leq x_R(u)$, and its bottom side on $\sigma(v)$ since $y_B(e) = y(v)$ and $x_L(v) \leq x_L(e) < x_R(e) \leq x_R(v)$. We will show that we can choose a vertical band $b'(e) \subset b(e)$ of non-zero width that has its top side on $\sigma(u)$ its bottom side on $\sigma(v)$ and does not intersect with any other vertex-segment $\sigma(a)$. For the coordinates of $b'(e)$ we choose $y'_B = y_B(e) = y(v)$, $y'_T = y_T(e) = y(u)$, and $x_L(e) < x'_L < x'_R < x_R(e)$. If there is a directed path in G from vertex a to vertex $orig(e)$ or a directed path from $dest(e)$ to a then $y(a) > y(u)$ or $y(v) > y(a)$. Therefore vertex segment $\sigma(a)$ and vertical band $b'(e)$ do not intersect. For this we assume that (see Lemma 3.4) G^* either has a directed path from $right(e)$ to $left(a)$ or has a directed path from $right(a)$ to $left(e)$. Then either $x'_R < x_R(e) \leq x_L(a)$ or $x_R(a) \leq x_L(e) < x'_L$ In both cases $b'(e)$ does not intersect with $\sigma(a)$.

Finally since the topological numbering X is optimal the vertex-segment of a non-sink vertex u is covered by the bottom sides of the maximal visibility bands of edges incident from u. Similarly the vertex-segment of a non-source vertex v is covered by the top sides of the maximal visibility bands of edges incident to v. Therefore vertex-segment $\sigma(v)$ is ε-visible from vertex-segment $\sigma(u)$ only if G has an edge (u, v).

The $O(n)$ time bound follows easily since each step of the algorithm can be accomplished in $O(n)$ time. □

3.5 DAGmaps and Directed Three-Dimensional ε-Visibility Representations

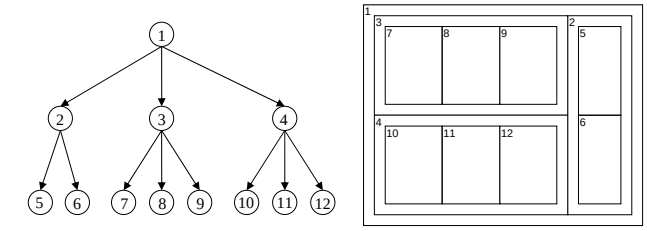
A treemap determines a directed three-dimensional ε-visibility representation of a tree T if vertex rectangles are placed in three dimensional space such that their x and y coordinates are unaltered and their z coordinates are equal to the tree height minus the distance of the corresponding vertices from the root (plus tiny perturbations in order to keep the rectangle disjoint) (see Fig. 3.4). The above discussion is extended to DAGmaps as described by Theorem 3.8.

Theorem 3.8 *When a DAG G admits a DAGmap then it admits a directed three-dimensional ε-visibility representation which can be constructed by shifting the vertex rectangles along the vertical direction in such a way that their z-coordinates are determined by an optimal topological numbering of G (plus a tiny perturbation).*

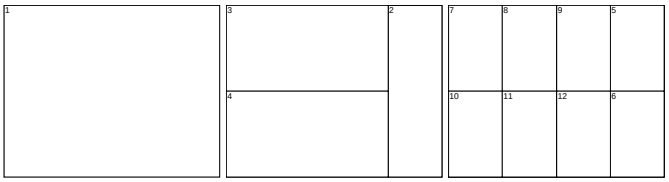
The proof of Theorem 3.8 is similar to the proof of Theorem 3.1.

Corollary 3.3 *A treemap determines a three-dimensional ε-visibility representation of a tree.*

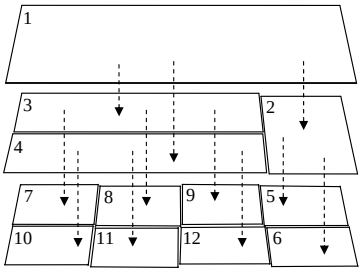
It is interesting to investigate if the converse of the Theorem 3.8 holds. If yes then DAGmap admissibility would be equivalent to three-dimensional visibility representation and results derived for the former problem would be valid for the second and vise versa. However the converse of Theorem 3.8 does not hold as the counter-example in Fig. 3.5 shows.



(a) A tree T (b) A treemap of T



(c) Layer 3 (d) Layer 2 (e) Layer 1



(f) Three-dimensional ε-visibility of T

Figure 3.4: A tree T, a treemap of T and the three-dimensional ε-visibility representation of T that is constructed from the treemap of T

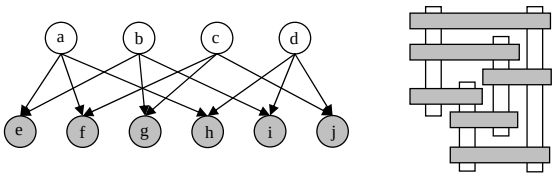


Figure 3.5: The DAG of this figure does not admit a DAGmap 2.7(a). However it admits a three-dimensional directed ε-visibility representation.

Chapter 4

DAGmaps and Dominance Relationships

In this chapter we show that vertex dominations in a DAG G correspond to rectangle inclusions in any DAGmap of G. Then based on that theorem we propose a heuristic algorithm that decompose a DAG G into component st-graphs, then draws each one separately and finally combines the results (see Fig. 4.1). If a component st-graph does not admit a DAGmap then the algorithm transforms it into a TTSP digraph by performing vertex duplications. Note that duplications inside a component st-graph do not propagate to the rest of DAG G. Our algorithm performs significantly fewer duplications than the transformation of G into a tree, provided that G admits a decomposition into component st-graphs. Such a decomposition can be found using dominance relationships among vertices. A nice feature of our algorithm is that it is independent of the layout function and for each component st-graph we can choose a different layout function. Especially when a component st-graph is TTSP then any layout function can be used, e.g., one that reduces the aspect ratios of the rectangles.

4.1 Preliminaries

A common characteristic of trees and TTSP digraphs is the existence of vertices that dominate all paths from the source vertex to a subset of vertices. In trees all paths from the root to vertices of a subtree are dominated by (i.e. pass from) the root of the subtree. In TTSP digraphs the source dominates all vertices and recursively the source of every composing subgraph dominates all vertices of the subgraph. Dominance was first introduced by Prosser in 1959 [159] and an algorithm for its computation was first presented by Lowry and Medlock ten years later [126]. The computation of dominators in DAGs and in general in directed graphs has applications in several areas, including program analysis and optimization [181, 142], circuit testing [14], distributed authorization [141], and theoretical biology [8, 9, 36]. We show that this problem provides a framework that allows for the clever separation of subgraphs in order to reduce the vertex duplications in DAGmaps.

A *flowgraph* $G = (V, E, s)$ is a directed graph with $n = |V|$ vertices and $m = |E|$ directed edges such that every vertex is reachable from a distinguished

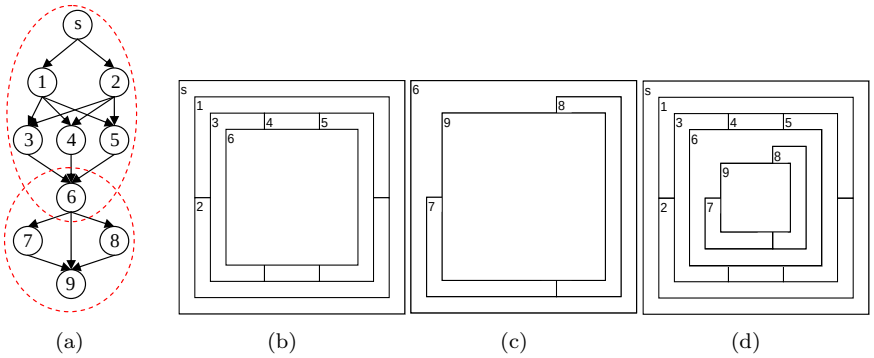


Figure 4.1: (a) A DAG which can be decomposed into two component st-Graphs. (b) DAGmap of the component st-Graph which is induced by the vertices 1, 2, 3, 4, 5 and 6. This subgraph is a MVSP digraph 2.5.2 and admits a DAGmap. (c) DAGmap of the component st-Graph which is induced by the vertices 6, 7, 8 and 9. This subgraph is a TTSP digraph. (d) DAGmap of G which is a combination of DAGmaps of (b) and (c).

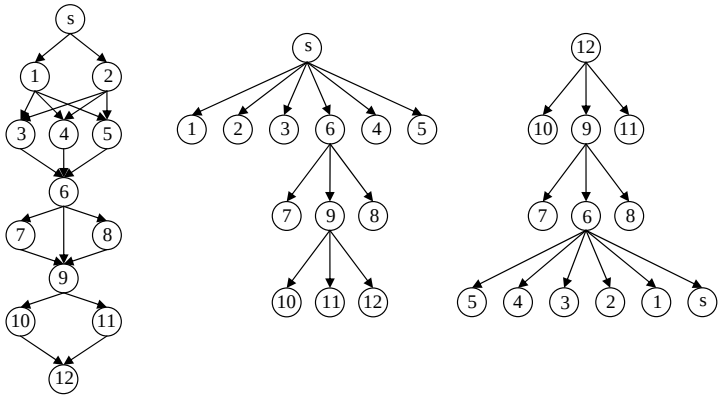
source vertex $s \in V$. In our study we consider acyclic flowgraphs, namely we assume that G is a single source Directed Acyclic Graph (DAG) and s is the source.

A vertex w *dominates* a vertex v if every path from s to v passes through w. Let $Dom(v)$ be the set of the vertices that dominate v. Obviously, s and v the trivial dominators of v, are in $Dom(v)$. Vertex w is the *immediate dominator* of v, denoted $w = \text{idom}(v)$, if w dominates v and every other dominator ($u \in Dom(v) \setminus \{v\}$) of v dominates w. Every vertex of a flowgraph $G = (V, E, s)$ except s has a unique immediate dominator. The edges $\{(\text{idom}(v), v) \mid v \in V \setminus \{s\}\}$ form a directed tree rooted at s, called the *dominator tree* of G, such that w dominates v if and only if w is an ancestor of v in the dominator tree [126, 3]. Therefore the set Dom(v) of vertices that dominate v are its ancestors in the dominator tree (see Fig. 4.2). The *post-dominators* of G are defined as the dominators in the graph obtained from G by reversing all directed edges. Then *post-idom*(v) denotes the immediate post-dominator of v and *post-Dom*(v) denotes the set of vertices that post-dominate v. In this setting G is assumed to contain a sink vertex t reachable from all $v \in V$. If DAG G has more than one sink vertices then t is introduced as an artificial vertex of G which is attached to all other sinks.

A component st-graph $G_{u,v}$ of G is a subgraph of G with a single source u and a single sink v that contains at least two edges and that is connected with the rest of G only through vertex u and/or vertex v (see Fig. 4.1).

4.2 Dominance and DAGmap Visualization

In DAGmaps of trees and TTSP digraphs the rectangle of a dominator u includes the rectangles of all vertices that it dominates. Similarly the rectangle of a post-dominator is equal to the union of the rectangles of vertices that it post-



(a) A DAG (b) Dominator tree (c) Post-dominator tree

Figure 4.2: An example showing a DAG and its dominator and post-dominator trees.

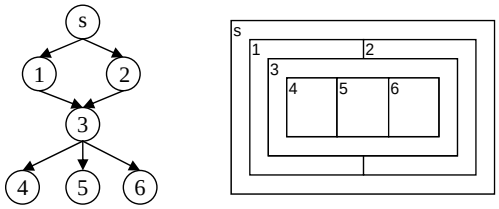


Figure 4.3: A counter-example that shows that the converse of Theorem 4.1 does not hold. From the DAG and its corresponding DAGmap we see that $R_4 \subset R_1$ but vertex 1 does not dominate vertex 4.

dominates. These observations hold for arbitrary DAGs as the following theorem shows.

Theorem 4.1 *Let $G = (V, E)$ be a DAG and u be a vertex of G. In a DAGmap of G the rectangle of vertex u includes the rectangles of all vertices that are dominated (resp. post-dominated) by u.*

Proof: Let v be a vertex that is dominated by u and $u = v_1, \ldots, v_k = v$ be a longest path of length k from u to v. From constraint B4 we have that $R_{v_2} = \cup_{e \in \Gamma^-(v_2)} R_e \subset \cup_{w \in N^-(v_2)} = R_u \Rightarrow R_{v_2} \subset R_u$. By induction on the length of the path and based on the fact that all paths to vertex v pass through vertex u we have that $R_{v_l} \subset \cup_{w \in N^-(v_l)} R_w \subset \cup_{w \in N^-(v_l)} R_u \Rightarrow R_{v_l} \subset R_u$ for every $l \in \{1, \ldots, k\}$. Therefore $R_v \subset R_u$.
The proof for the case that u is post-dominated by vertex v is similar. □

The counter-example of Fig. 4.3 shows that the converse of Theorem 4.1 does not hold.

In the DAG of Fig. 4.2(a) vertex s dominates vertex 6 and vertex 6 post-dominates vertex s therefore $R_s = R_6$. The same holds for the pairs of vertices

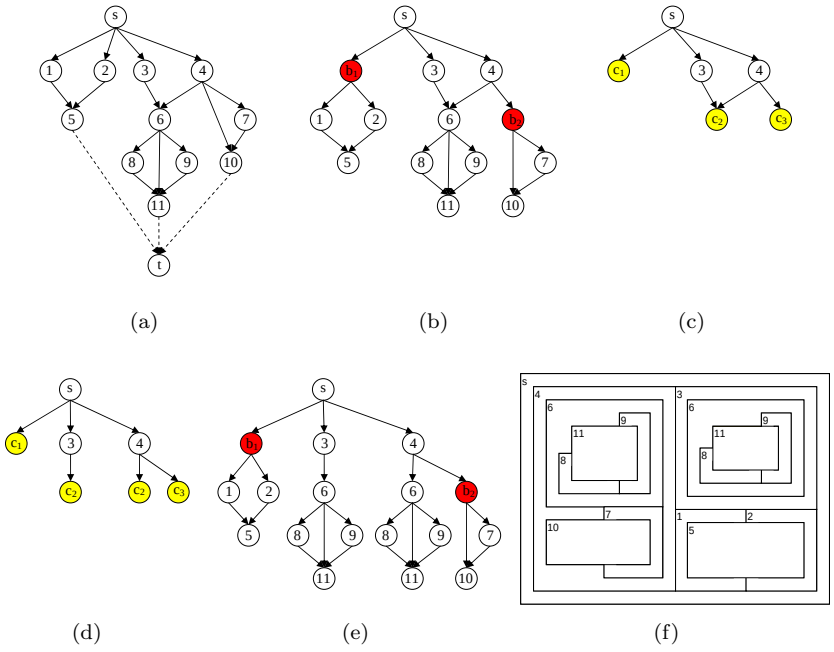


Figure 4.4: An example of the transformation of a DAG G into G' that admits a DAGmap. a) A DAG with an artificial sink vertex t. b) Introduction of artificial b-vertices. c) Folding of st-subgraphs into c-vertices. d) Vertex duplications. e) Unfolding of c-vertices; the component st-graph $G_{6,11}$ appears twice in DAG G'. f) DAGmap drawing of G'.

$s, 9$ and $s, 12$. Therefore $R_s = R_6 = R_9 = R_{12}$. Another observation is that the rectangle of a vertex includes the rectangles of the vertices that it dominates (e.g., $R_7 \subset R_6$, $R_8 \subset R_6$ and $R_9 \subset R_6$).

From Theorem 4.1 it follows that if u dominates v and v post-dominates u then $R_u = R_v$. The equality of rectangles R_u and R_v can be used as a heuristic rule to reduce the number of vertex duplications that are needed to convert a DAG G into a DAG G' that admits a DAGmap drawing. This is because duplications of vertices different from u and v that appear in paths from u to v do not trigger duplications in the out-neighbors of v (i.e. the duplications stop at v). The rules for vertex duplications that we propose are:

Definition 4.1 (Heuristic Rules for Vertex Duplications)

a) If u is duplicated and $G_{u,v}$ is a component st-graph then duplicate the whole $G_{u,v}$.

b) The propagation of duplications of a vertex w of a component st-graph $G_{u,v}$, where $w \neq u, v$, stops at vertex v.

4.2.1 Separating the subgraphs of a DAG

Before we apply the duplication rules of Definition 4.1 it is recommended to separate the subgraphs of G that have the property that the source dominates and the sink post-dominates the remaining vertices. For example in Fig. 4.4 vertex s dominates vertex 5 but vertex 5 does not post-dominate vertex s. This technical problem is solved by the introduction of an artificial vertex, b_1, which we call a *b-vertex*. On the other hand the subgraph induced by vertices $s, 3, 4$ and 6 cannot be separated since vertex 4 has an out-going edge to a vertex that does not belong to the subgraph. In order to guarantee the admissibility of a DAGmap drawing and taking into account that vertex 6 has two in-coming edges, vertex 6 is duplicated and according to duplication heuristic rules the whole component st-graph $G_{6,11}$ is duplicated.

Algorithm 10 outlines the main steps to introduce b-vertices. List *pdoms* contains all non-leaves of the post-dominator tree of DAG G. The check whether u post-dominates w can be done efficiently using the post-dominator tree. Without loss of generality we assume that the input is an st-graph.

Algorithm 10 Introduction of b-vertices

Input: st-graph $G = (V, E)$ and list *pdoms* of all non-leaves of the post-dominator tree of G
Output: st-graph G' in which b-vertices have been introduced
Uses: List L of vertices

```
for each vertex u ∈ pdoms do
    v ← idom(u);
    if (post-idom(v) ≠ u) then // post-idom(v): immediate post-dominator of v
        L ← ∅;
        for each vertex w ∈ N⁺(v) do
            if (w ∈ post-Dom(u)) then
                L.add(w);
        if (list L contains more than one vertex) then
            Introduce a b-vertex b
            Add edge (v, b)
            Add edges from b to all vertices in L
            Remove edges from v to all vertices in L
            Update idom and post-idom of the affected vertices
```

4.2.2 Duplication Algorithm

To simplify the duplication algorithm we fold (hide) all vertices that belong to a component st-graph of G with into an artificial cluster vertex or c-vertex. Then we obtain a DAG with normal and c-vertices where each c-vertex may recursively contain other c-vertices. DAG G itself is folded into a c-vertex. Algorithm 11 outlines the steps to convert, via vertex duplications, DAG G into a DAG G' that admits a DAGmap. Duplications are restricted inside a component st-graph.

Algorithm 11 Vertex Duplication Algorithm

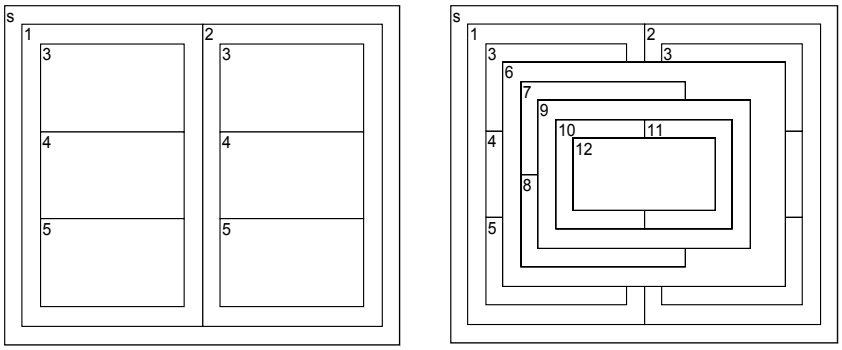
Input: st-graph $G = (V, E)$, list of post-dominators *pdoms* and array *layer* indexed by vertices that stores the layer of vertices of G
Output: st-graph G' which admits a DAGmap
Uses: Queue Q of c-vertices initially empty and list L of pairs of vertices

```
function foldSubgraphs(G = (V, E), pdoms)
    for each vertex u ∈ pdoms do
        if (post-idom(idom(u)) = u) then
            L.add(idom(u), u);
    Sort L in ascending order according to layer[u] − layer[v], where (u, v) ∈ L
    for each (u, v) ∈ L do
        Fold (hide) subgraph G_{u,v} inside an artificial c-vertex

function duplicateVertices(cvertex)
    Q.enqueue(cvertex);
    while (Q ≠ ∅) do
        u ← Q.dequeue();
        Unfold vertex u to get DAG G_1 = (V_1, E_1);
        if (DAG G_1 does not admit a DAGmap) then // convert G_1 into TTSP
            Let t be the sink vertex of G_1;
            Convert the DAG induced by V_1 \ {t} into a tree T;
            Add edges from all leaves of T to vertex t to get DAG G'_1;
            for each c-vertex or copy of c-vertex v of G'_1 do
                Q.enqueue(v);
```

Theorem 4.2 *If we change Algorithm 11 so that it converts G_1 into a TTSP digraph even in the case that G_1 admits a DAGmap then the produced st-graph G' is TTSP.*

Proof: When G_1 is subgraph of G_2 then Algorithm 11 first folds G_1 and then folds G_2. The algorithm unfolds G_1 and G_2 in reverse order; first G_2 and then G_1. Therefore duplications of vertices of G_1 do not affect vertices of G_2 that do not belong to G_1. By induction on recursive unfolding of c-vertices we prove that G' is TTSP as follows: a) The first c-vertex that hides the whole graph is not TTSP but we unfold it and then the st-graph minus the sink vertex is converted into a tree T. Moreover the sink t is adjacent to all leaves of T. Therefore the resulting graph is a TTSP digraph. b) Let u be a c-vertex of a TTSP digraph G_2. There are three cases concerning the position of vertex u in G_2: 1) it is the source of G_2, 2) it is the sink of G_2 and 3) there are two serially connected subgraphs G_3 and G_4 of G_2 and vertex u is the sink of G_3 and the source of G_4. When we unfold vertex u we get a subgraph G_1 which we transform it into a DAG G'_1 which by the induction hypothesis is a TTSP digraph. Graph G'_1 replaces vertex u in G_2 and depending on the position of u in G_2 we have the following cases for the transformed DAG G'_2: 1) G'_2 is a series composition of G'_1 and G_2 where the sink of G'_1 is identified with the source of



(a) (b)

Figure 4.5: DAG G of Fig. 4.2(a) is converted into a TTSP digraph G' by duplications of vertices 3, 4 and 5 and then a DAGmap of G' is drawn. a) Only the first three layers from the source are shown. b) The whole hierarchy is shown.

G_2, 2) G'_2 is a series composition of G_2 and G'_1 where the sink of G_2 is identified with the source of G'_1 and 3) G'_2 contains a series composition of G_3 and G'_1 and G_4 where the sink of G_3 is identified with the source of G'_1 and the sink of G'_1 is identified with the source of G_4. In all three cases G'_2 is a TTSP digraph. □

The DAG of Fig. 4.2(a) consists of 13 vertices. If we convert it into a tree, the resulting tree has 117 vertices. If we duplicate vertices 3, 4 and 5 of the subgraph $G_{s,6}$ which is induced by vertices s,1, 2, 3, 4, 5 and 6 the resulting DAG, which is a TTSP digraph, has 16 vertices since the propagation of duplications stops at vertex 6. Note that subgraph $G_{s,6}$ contains a $K_{2,3}$ DAG which admits a DAGmap (see Sec. 2.5.2). Therefore DAG G admits a DAGmap without any duplication.

4.2.3 DAGmap Algorithm

Let G be a DAG that admits a DAGmap. DAG G might be the output of Algorithm 11. As in Algorithm 11 the component st-graphs of G are folded inside artificial c-vertices. Note that G itself is also folded into an artificial c-vertex. The algorithm that we propose 12 takes as input an artificial c-vertex, unfolds it, calculates rectangles for the vertices of the corresponding st-graph, and then recursively calls itself for each artificial c-vertex that is contained in the st-graph. Finally, the vertices of G are sorted in ascending order according to their longest path distance from the source and then the rectangles of the vertices of G are drawn. In Algorithm 12 we assume that each st-graph, that is unfolded, admits a DAGmap and that we have an algorithm to draw such a DAGmap.

4.2.4 Finding Dominators

For completeness of our discussion we will briefly review the problem of finding the immediate dominators of vertices in a flowgraph and we will describe a

Algorithm 12 DAGmap Algorithm

Input: DAG G that admits a DAGmap, list of post-dominators *pdoms* and rectangle R
Output: DAGmap of G
Uses: Artificial vertex c

```
c ← foldSubgraphs(G, pdoms);
dagmapRecursive(c, R);
Draw the rectangles of vertices of G;

function foldSubgraphs(G = (V, E), pdoms)
    for each vertex u ∈ pdoms do
        if (post-idom(idom(u)) = u) then
            L.add(idom(u), u);
    Sort L in ascending order according to layer[u] − layer[v], where (u, v) ∈ L
    for each (u, v) ∈ L do
        Fold (hide) subgraph G_{u,v} inside an artificial c-vertex

function dagmapRecursive(c, R)
    Unfold vertex c to get DAG G_1 = (V_1, E_1); //DAG G_1 admits a DAGmap
    Calculate a DAGmap of G_1 on R;
    for each artificial c-vertex u ∈ V_1 do
        dagmapRecursive(u, R_u);
```

simple and practical algorithm that works fast on DAGs. Lorry and Medlock introduced an $O(n^4)$ algorithm to find all the immediate dominators in a flow-graph. Successive improvements to this time bound were achieved leading to a practical $O(m\alpha(m,n))$-time algorithm [120], where α is the inverse of Ackermann function and grows extremely slow with m and n. Harel [88] claimed a linear-time algorithm, but Alstrup et al. [12] found problems with some of his arguments and developed a corrected algorithm, which uses powerful bit-manipulation-based data structures that make any implementation impractical. Buchsbaum et al. [50, 51] proposed a simpler algorithm. Georgiadis and Tarjan [81] gave a linear-time algorithm for the pointer machine computation model while the previously mentioned linear time algorithms are for the random-access model of computation. For more complete discussion about the performance of dominance algorithms see Georgiadis et al. [82].

We will describe a simple iterative algorithm initially presented by Allen and Cocke and later refined by Cooper, Harvey and Kennedy. The refined algorithm runs in time $O(n^2 + m)$. Allen, in her work on control-flow analysis, formulated the dominance computation as a global data-flow problem [7]. The sets $\mathrm{Dom}(v)$ are the unique maximal solution to the following data-flow equations:

$$\mathrm{Dom}'(v) = \left(\cap_{u \in N^-(v)} \mathrm{Dom}'(u)\right) \cup \{v\} \ , \quad \forall v \in V \tag{4.1}$$

In a 1972 paper, Allen and Cocke showed an iterative algorithm to solve these equations [6] by initializing

$$dom'(v) \leftarrow \begin{cases} \{s\}, & v = s \\ V, & \text{otherwise} \end{cases}$$

and repeatedly applying the following step until it no longer applies:

> Find a vertex v such that Eq. 4.1 is false and replace $\mathrm{Dom}'(v)$ by the expression on the right side of (4.1).

To improve the performance of this simple algorithm, Cooper et al. [58] proposed to keep *Dom* sets ordered and to represent them as paths in dominance tree T. The dominance tree itself is represented by a single array indexed by vertices. Let *parent* denote this array. For each vertex $u \in V \setminus \{s\}$ the entry $parent[u]$ is the parent of u in tree T and the immediate dominator of u in graph G. With the above representation set $Dom(u)$ is written as: $Dom(u) = \{s\} \cup \ldots \cup \{parent[parent[u]]\} \cup \{parent[u]\} \cup \{u\}$.

We recall here that the *Dom* sets are ordered and reflect paths in the dominance tree. With this order, if $Dom(u) \cap Dom(v) \neq \emptyset$ then the resulting set is a prefix of both $Dom(u)$ and $Dom(v)$. Therefore intersection is performed by starting at the back of the *Dom* sets and moving toward the front comparing elements until they are the same.

Algorithm 13 is a simplified version of the algorithm of Cooper et al. [58] that exploits the fact that the input is a DAG and relies on the topological sorting of vertices to avoid repeated iterations. The algorithm runs in worst case time $O(n + m \cdot h)$, where h is the size of the largest $Dom(v),\ v \in V$ set which is equal to the height of the dominance tree [58, 82]. This time bound is based on the fact that the algorithm performs $O(m)$ intersections each of which can take time equal to the height of the dominance tree. However in practice the average time for an intersection is much smaller than h. Vertices are ordered in reverse postorder which in DAGs is also a topological order. Each vertex v is assigned the number $|V| - postorder\ number(v)$.

Algorithm 13 Simple Dominance Algorithm

Input: single source DAG $G = (V, E)$ and source vertex s
Output: array *parent* that represents the dominance tree

```
function dominance(G = (V, E), s)
    for each u ∈ V do parent[u] ← UNDEFINED;
    D ← DFS(s), parent[s] = s;
    for each v ∈ V \ {s} in reverse postorder of D do
        x ← UNDEFINED;
        for each u ∈ N⁻(v) do
            if x = UNDEFINED then x ← u; else x ← intersect(x, u);
        parent[v] ← x; // x is the immediate dominator of v

function int intersect(x, y)
    while x ≠ y do
        while x > y do
            x ← parent[x];
        while y > x do
            y ← parent[y];
    return x;
```

Chapter 5

Technical Issues and Software for DAGmaps

This chapter is devoted to technical issues on space filling visualization of DAGs. First the issue of hierarchy presentation is described and then the program DAGmap View is demonstrated.

5.1 Hierarchy Presentation

A shortcoming of treemaps is that the structure of the hierarchy is not easily distinguishable. For these reason many techniques have been proposed to enhance the hierarchy presentation. Three well known techniques are the cushion [213], the nested [183], and the cascaded [127] presentations. The cushion presentation uses shading as a strong extra cue to emphasize the hierarchical structure. It is appealing but in terms of usefulness is inferior to the other two methods. It consumes processing resources and does not improve significantly the presentation of the structure of a tree. Moreover its applicability to DAGmaps is very questionable. The cascaded presentation, which has been recently introduced, shifts the rectangles of children down and to the right relative to their parent's rectangle. It creates a depth effect resembling a stack of cards which looks nice for treemaps but when used in DAGmaps it causes confusion. When the rectangle R_v of a vertex v is drawn on the union of rectangles of its ancestors then by shifting R_v down and to the right it may hide some of its ancestors' rectangles. Even worse R_v may overlap with a rectangle R_u although there is no path from v to u or from u to v. From the three hierarchy presentation techniques which are mentioned above, nesting is the only one that can be applied to DAGmaps without causing confusion. For example, nesting does not change the property that a rectangle R_v overlaps with a rectangle R_u only when there is a path either from v to u or from u to v. However the problem that a rectangle may hide some of its ancestors' rectangles still exists with nesting although it is less severe than it is with cascading.

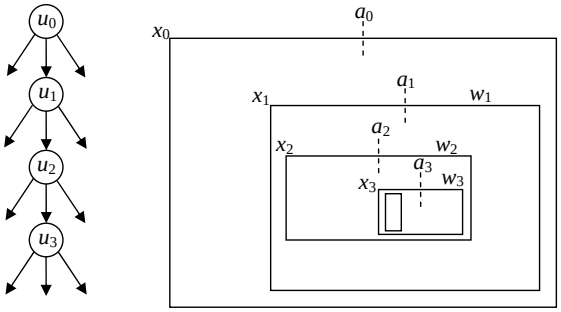
5.2 Nesting

In treemaps, nesting is trivial and is done along with the layout. The drawing rectangle R_u of a node u is shrunk and the resulting rectangle R'_u is located inside R_u. Then the border $R_u \setminus R'_u$ is used for displaying information concerning u and R'_u is used for drawing the children of u (see Fig. 5.2). And this procedure is repeated recursively. In DAGmaps, layout and nesting should be implemented as two separate functions. First, the layout function assigns rectangles to vertices and/or to edges of a DAG G, in case that such an assignment is possible. Then the nesting function shrinks the rectangles according to their longest path distance from the sources, in order to reveal the hierarchy structure. Decomposing the nesting from the layout greatly facilitates the implementation of the layout algorithm although slightly complicates the implementation of the nesting algorithm. Additionally the decomposition of the two functions allows precalculation of the coordinates of the rectangles of the vertices of G and navigation through the hierarchy without changing the mental map of the user. Note that the squarified layout 3 is unstable [30] and a small change in the aspect ratio of a rectangle R has a dramatic effect on the position of the rectangles in which R is partitioned. One minor drawback of the decomposition of nesting from layout is that the aspect ratio of the displayed rectangles is slightly higher than the aspect ratio of the rectangles produced by the combined nesting and layout. This happens because the layout function decides how to partition the original rectangle and not the transformed image rectangle which is actually displayed. In general the transformation of a rectangle is not a uniform scaling and the image rectangle may have different aspect ratio than the original rectangle. Therefore the decisions of the layout function are not the best possible. On the other hand the combined nesting and layout function decides how to partition the rectangles that are actually displayed. Therefore the decisions are the best possible that the partitioning algorithm can achieve.

5.2.1 Nesting in Treemaps

We will present the nesting function in the case of a tree. Suppose we have a tree with root u_0 and assume that $u_0, u_1, \ldots, u_h$ is a path from root u_0 to a leaf u_h. Let R_{u_i} be the rectangle assigned to node u_i, $i = 0, 1, 2, \ldots, h$ after the recursive calls of the layout function (Fig. 5.1). Assume that R_{u_i} is represented by (x_i, y_i, w_i, h_i), where x_i and y_i are the horizontal and vertical coordinates of the upper left corner of the rectangle, and w_i and h_i are the width and height of the rectangle respectively. Each of rectangle R_{u_i}, $i \in \{0, 1, \ldots, h-1\}$ is partitioned into rectangles representing the children of node u_i. From Proposition 2.1 we have that rectangle R_{u_i} is equal to the union of the rectangles of the children of u_i ($R_{u_i} = \cup_{v \in children(u_i)} R_v$). The nesting function shrinks rectangle $\cup_{v \in children(u_i)} R_v$ to uncover a border of R_{u_i}. For aesthetic reasons we assume that the distance of each side of the image rectangle is d from the corresponding side of R_{u_i} (Fig. 5.2). This requirement leads to different horizontal and vertical scalings.
In the following we describe the transformation of rectangles $R_{u_1}, \ldots, R_{u_h}$ by the nesting function. We consider only the horizontal direction, since the analysis in the vertical direction is similar.

First consider that only rectangle $\cup_{v \in children(u_i)} R_v$ is transformed. If w_i



(a) A path (b) Corresponding rectangles

Figure 5.1: A path P in a tree and the rectangles that correspond to nodes of P.

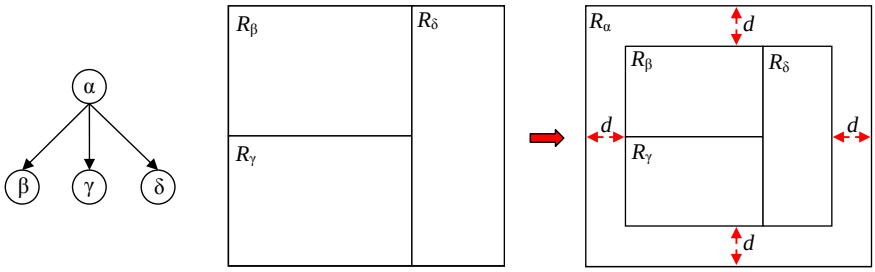


Figure 5.2: Rectangle R_α is partitioned into rectangles R_β, R_γ and R_δ. Then rectangle $R_\beta \cup R_\gamma \cup R_\delta$ is shrunk to reveal a border of rectangle R_α. The distance of each side of the shrunk $R_\beta \cup R_\gamma \cup R_\delta$ from the corresponding side of R_α is d.

and w_i^1 is the width of $\cup_{v \in children(u_i)} R_v$ before and after the transformation we have:

$$\frac{w_i^1}{w_i} = \frac{w_i - 2d}{w_i} = 1 - \frac{2d}{w_i} \tag{5.1}$$

Now let a_i be a point of interval $(x_i, x_i + w_i)$ such that the shrinking of rectangle $\cup_{v \in children(u_i)} R_v$ in the horizontal direction is done towards line $x = a_i$. A reasonable choice for a_i is $a_i = x_i + \frac{w_i}{2}$. Also let x be the horizontal coordinate of a point $P \in \cup_{v \in children(u_i)} R_v$ before the transformation and x^1 be the horizontal coordinate of point P after the transformation. Then we have:

$$x^1 - a_i = \frac{w_i^1}{w_i}(x - a_i) = (1 - \frac{2d}{w_i})(x - a_i) \Rightarrow$$

$$x^1 = x + 2d\frac{a_i - x}{w_i} \tag{5.2}$$

When rectangle R_{u_i} is transformed all rectangles $R_{u_{i+1}}, \ldots, R_{u_h}$ of the descendant nodes of u_i are transformed (Fig. 5.1).

Now consider that the nesting function transforms rectangles $R_{u_1}, \ldots, R_{u_h}$ one after the other in this order. The final position and dimensions of a rectangle

R_{u_k} are affected by the transformation of all rectangles of nodes in the path from root u_0 to node u_k.

Lemma 5.1 *Let w_i be the initial width of rectangle R_{u_i}, $i \in \{1, \ldots, k\}$ and w_k^l be the width of rectangle R_{u_k} after the explicit transformation of rectangles $R_1, R_2, \ldots, R_l$, $l < k$ by the nesting function. Then width w_k^l is given by the following equation:*

$$w_k^l = w_k(1 - 2d\sum_{i=1}^{l}\frac{1}{w_i}) \tag{5.3}$$

Proof: We will prove (5.3) using induction on l.
For $l = 1$ from (5.1) we have $w_k^1 = w_k(1 - 2d\sum_{i=1}^{1}\frac{1}{w_i})$.
We assume that (5.3) holds for $l \leq l'$ and we will prove that (5.3) holds for $l = l' + 1$.
If $w_k^{l'}$ is the width of rectangle R_k after the explicit transformation of rectangles $R_1, R_2, \ldots, R_{l'}$, in this order, and $w_k^{l'+1}$ the width of R_k after the explicit transformation of $R_{l'+1}$, we have:

$$w_k^{l'+1} = w_k^{l'}\frac{w_{l'+1}^{l'+1}}{w_{l'+1}^{l'}} \Rightarrow w_k^{l'+1} = w_k^{l'}(\frac{w_{l'+1}^{l'} - 2d}{w_{l'+1}^{l'}})$$

where $w_{l'+1}^{l'}$ is the width of $R_{l'+1}$ after the explicit transformation of rectangles $R_1, R_2, \ldots, R_{l'}$, and $w_{l'+1}^{l'+1}$ the width of $R_{l'+1}$ after its own explicit transformation. By the induction hypothesis, we have:

$$w_k^{l'+1} = w_k(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i})\frac{w_{l'+1}(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i}) - 2d}{w_{l'+1}(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i})} \Rightarrow$$

$$w_k^{l'+1} = w_k\frac{w_{l'+1}(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i}) - 2d}{w_{l'+1}} \Rightarrow$$

$$w_k^{l'+1} = w_k(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i} - \frac{2d}{w_{l'+1}}) \Rightarrow w_k^{l'+1} = w_k(1 - 2d\sum_{i=1}^{l'+1}\frac{1}{w_i})$$

□

Similarly the following lemma describes the transformation of horizontal coordinates.

Lemma 5.2 *Let x_i be the initial horizontal coordinate of the upper left corner of rectangle R_{u_i}, $i \in \{1, \ldots, k\}$. Let w_i be the initial width of rectangle R_{u_i} and a_i be a point of interval $(x_i, x_i + w_i)$ such that the shrinking of rectangle $\cup_{v \in children(u_i)} R_v$ in the horizontal direction is done towards line $x = a_i$, $i \in \{1, \ldots, k\}$. Let x is the horizontal coordinate of a point P of the rectangle R_k and x^l be the horizontal coordinate of P after the explicit transformation of rectangles $R_1, R_2, \ldots, R_l$, $l < k$ by the nesting function. We assume that l and d are sufficiently small so that width $w_k^l > 0$ (see Lemma 5.1). Then coordinate x^l is given by the following equation:*

$$x^l = x + 2d\sum_{i=1}^{l}\frac{a_i - x}{w_i} \tag{5.4}$$

Proof: The proof will be done by induction on l.
For $l = 1$ from (5.2) we have: $x^1 = x + 2d\frac{a_1 - x}{w_1}$.
We assume that (5.4) holds for $l \leq l'$.
We will show that (5.4) holds for $l = l' + 1$.
If $x^{l'}$ is the horizontal coordinate of the point P after the transformation of rectangles $R_1, R_2, \ldots, R_{l'}$, in this order, and $x^{l'+1}$ if the horizontal coordinate of P after the explicit transformation of rectangle $R_{l'+1}$, then using (5.2) we have:

$$x^{l'+1} = x^{l'} + 2d\frac{a^{l'}_{l'+1} - x^{l'}}{w^{l'}_{l'+1}} \tag{5.5}$$

where $x = a^{l'}_{l'+1}$ is the equation of the line toward which the rectangle $R_{l'+1}$ shrinks in the horizontal direction, and $w^{l'}_{l'+1}$ is the width of rectangle $R_{l'+1}$ after the transformation of rectangles $R_1, R_2, \ldots, R_{l'}$. Applying the induction hypothesis for $a^{l'}_{l'+1}$ and $x^{l'}$ and (5.3) for $w^{l'}_{l'+1}$ on the term

$$2d\frac{a^{l'}_{l'+1} - x^{l'}}{w^{l'}_{l'+1}} \ ,$$

we have:

$$2d\frac{a_{l'+1} + 2d\sum_{i=1}^{l'}\frac{a_i - a_{l'+1}}{w_i} - x - 2d\sum_{i=1}^{l'}\frac{a_i - x}{w_i}}{w_{l'+1}(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i})} =$$

$$2d\frac{a_{l'+1} - x - 2d(a_{l'+1} - x)\sum_{i=1}^{l'}\frac{1}{w_i}}{w_{l'+1}(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i})} =$$

$$2d(a_{l'+1} - x)\frac{1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i}}{w_{l'+1}(1 - 2d\sum_{i=1}^{l'}\frac{1}{w_i})} = 2d\frac{a_{l'+1} - x}{w_{l'+1}}$$

Using the above result and the induction hypothesis for $x^{l'}$, (5.5) becomes:

$$x^{l'+1} = x + 2d\sum_{i=1}^{l'}\frac{a_i - x}{w_i} + 2d\frac{a_{l'+1} - x}{w_{l'+1}} \Rightarrow$$

$$x^{l'+1} = x + 2d\sum_{i=1}^{l'+1}\frac{a_i - x}{w_i}$$

□

Based on (5.3) and (5.4) and setting $\Sigma_{1x} = \sum_{i=1}^{l}\frac{a_{xi}}{w_i}$, $\Sigma_{2x} = \sum_{i=1}^{l}\frac{1}{w_i}$, $\Sigma_{1y} = \sum_{i=1}^{l}\frac{a_{yi}}{h_i}$ and $\Sigma_{2y} = \sum_{i=1}^{l}\frac{1}{h_i}$ we can calculate nesting using Alg. 14

The nodes that are not assigned nesting rectangles are not displayed since there is not enough room in their parents. These nodes are assigned transformed rectangles when they become visible as the user navigates through the hierarchy.

Algorithm 14 Tree Nesting

Input: Rooted tree T where each node v has been assigned a rectangle $v.r$
Output: Some nodes are assigned transformed rectangles $nestedR$
Parameters: Border width d_x and height d_y

$root.nestedR = root.r$;
calculateNesting($root,\ 0,\ 0,\ 0,\ 0$);

function calculateNesting($v,\ \Sigma_{1x},\ \Sigma_{2x},\ \Sigma_{1y},\ \Sigma_{2y}$)
$x \leftarrow v.r.x$; $y \leftarrow v.r.y$; $w \leftarrow v.r.w$; $h \leftarrow v.r.h$;
$a_x \leftarrow x + w/2$; $a_y \leftarrow y + h/2$;
$\Sigma_{1x} \leftarrow \Sigma_{1x} + a_x/w$; $\Sigma_{2x} \leftarrow \Sigma_{2x} + 1/w$;
$\Sigma_{1y} \leftarrow \Sigma_{1y} + a_y/h$; $\Sigma_{2y} \leftarrow \Sigma_{2y} + 1/h$;
for $u \in v.children()$ **do**
if $((1 - 2d_x \cdot \Sigma_{2x} > 0)$ **and** $(1 - 2d_y \cdot \Sigma_{2y} > 0))$ **then**
$x_c \leftarrow u.r.x$; $y_c \leftarrow u.r.y$; $w_c \leftarrow u.r.w$; $h_c \leftarrow u.r.h$;
$u.nestedR.x \leftarrow x_c + 2d_x \cdot (\Sigma_{1x} - x \cdot \Sigma_{2x})$;
$u.nestedR.w \leftarrow w_c \cdot (1 - 2d_x \cdot \Sigma_{2x})$;
$u.nestedR.y \leftarrow y_c + 2d_y \cdot (\Sigma_{1y} - y \cdot \Sigma_{2y})$;
$u.nestedR.h \leftarrow h_c \cdot (1 - 2d_y \cdot \Sigma_{2y})$;
calculateNesting($u,\ \Sigma_{1x},\ \Sigma_{2x},\ \Sigma_{1y},\ \Sigma_{2y}$);

5.2.2 Nesting in TTSP digraphs

To show the hierarchy structure in DAGmaps of TTSP digraphs we use nesting which is calculated using an algorithm similar to Algorithm 14. The nesting algorithm takes as input a TTSP digraph G and its decomposition tree T. Similar to the case of trees, the transformations of the rectangles are based on (5.1) and (5.2). However it is not easy to express the recursive transformation of rectangles in a closed form. We recall here that the recursive transformation of rectangles in the case of trees is described by (5.3) and (5.4).

Suppose that DAG G consists of series composition of DAGs G_1 and G_2 where the sink of G_1 is identified with the source of G_2. The decomposition tree T of G has root an S-node whose left subtree is the decomposition tree of G_1 and whose right subtree is the decomposition tree of G_2 (see Fig. 5.3). The nesting algorithm traverses every S-node of T in inorder fashion. First it traverses the left subtree of an S-node then it transforms the rectangle of the S-node and then it traverses the right subtree of the S-node. It performs one shrinking transformation every time a right subtree of an S-node is followed. The shrinking transformations accumulate. The accumulated transformations of the left subtree of an S-node are carried to the S-node itself and to the right subtree.

In P-nodes the algorithm traverses the subtrees with an arbitrary order. Contrary to the S-node case, the accumulated transformations of one subtree are not carried to P-node or to other subtrees of the P-node. The only complication, in this case, is in the treatment of the nesting of the rectangle of the sink vertex t_P of the subgraphs of G whose composition is encoded by the P-node. Sink vertex t_P is encoded in the decomposition tree by an S-node which is the first

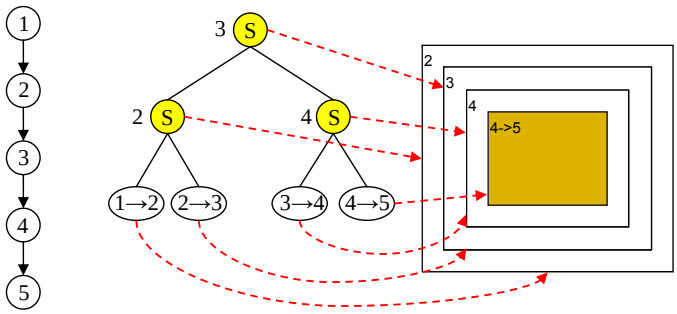


Figure 5.3: An example of how the nesting algorithm behaves on S-nodes of the decomposition tree T.

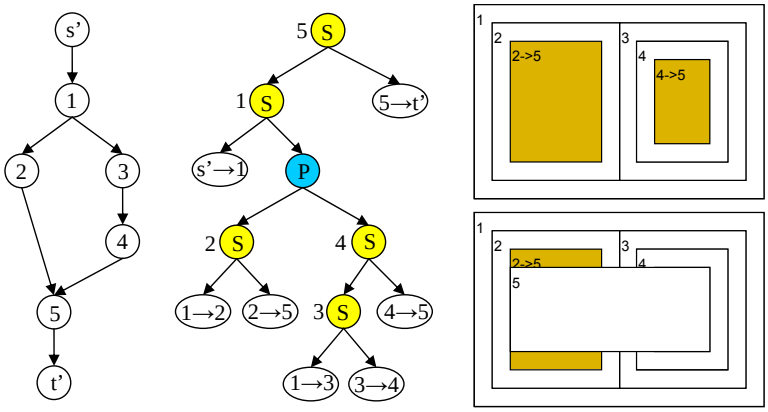


Figure 5.4: An example of how the nesting algorithm behaves on P-nodes of the decomposition tree T.

ancestor of the P-node that has this P-node on its left subtree (see Fig. 5.6 for examples). The borders of transformed rectangle R'_{t_P} are determined by the borders of the rectangles of edges in $\Gamma^-(t_P)$ (see Fig. 5.4). Then the coordinates of rectangle R'_{t_P} are used to solve the inverse linear system and determine what the values of $\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}$ should have been in order that the initial rectangle R_{t_P} of vertex t_P is transformed into rectangle R'_{t_P}. Finally the rectangles R'_e of edges $e \in \Gamma^-(t_P)$ are replaced by rectangles $R'_e \cap R'_{t_P}$ in order that rectangle R'_{t_P} to cover rectangle R'_e. An example of the inverse transformation is:

$$\begin{cases} x^l = x + 2d_x\Sigma_{1x} - 2d_x x\Sigma_{2x} \\ w^l = w(1 - 2d_x\Sigma_{2x}) \end{cases} \Rightarrow$$

$$\begin{cases} \Sigma_{2x} = (w - w^l)/(2d_x w) \\ \Sigma_{1x} = (x^l - x)/(2d_x) + x\Sigma_{2x} \end{cases} \tag{5.6}$$

In the decomposition tree T of a TTSP digraph G there is no explicit rep-

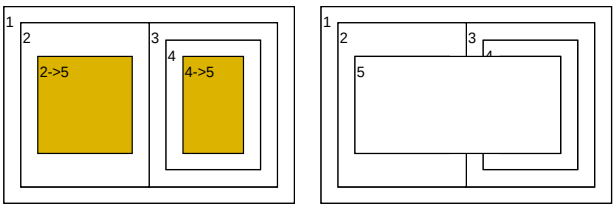
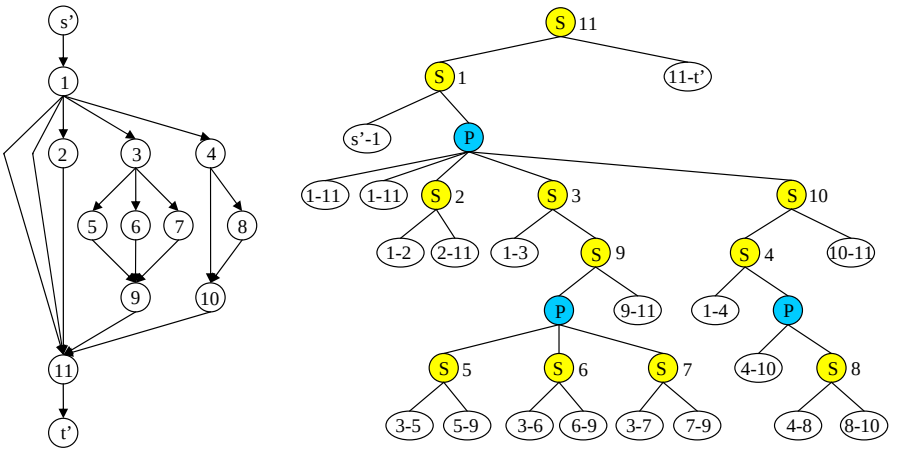


Figure 5.5: After the post processing step the rectangle of edge $(2, 5)$ is covered by the rectangle of vertex 5.



(a) DAG G' that corresponds to DAG G of Fig. 2.15(a)

(b) The decomposition tree of G'

Figure 5.6: An example of the construction of graph G'. Each node of G is represented by an S-node in the decomposition tree T' of G'.

resentation for the source and the sink vertices. To simplify the nesting algorithm we propose to substitute G with a TTSP digraph $G' = (V \cup \{s', t'\}, E \cup \{(s', s), (t, t')\})$, where DAG G' is constructed from G with the addition of a new source vertex s' that leads to the source vertex s of G and the addition of a new sink vertex t' that is adjacent to the sink vertex t of G. For an example see Fig. 5.6 where we apply the above construction to DAG G of Fig. 2.15(a).

In the post processing step each rectangle R'_e of edge $e \in E$ is replaced by rectangle $R'_e \cap R'_{dest(e)}$ (i.e. $R''_e \leftarrow R'_e \cap R'_{dest(e)}$), so that $R''_e \subset R'_{dest(e)}$ (see Fig. 5.4 and 5.5).

function restrictEdgeRectangles($G = (V, E)$)
  for each $e \in E$ **do**
    $e.nestedR \leftarrow e.nestedR \cap dest(e).nestedR$;

Algorithm 15 TTSP Nesting

Input: a) Decomposition tree T of a TTSP digraph G where each node v of T has been assigned a rectangle $v.r$ and b) the graph G
Output: Each node v is assigned a transformed rectangle $v.nestedR$ of $v.r$

$root.nestedR \leftarrow root.r$;
calcNesting($root,\ 0,\ 0,\ 0,\ 0$); // recursive call
restrictEdgeRectangles(G); // post processing

$[\Sigma'_{1x}, \Sigma'_{2x}, \Sigma'_{1y}, \Sigma'_{2y}, nr] \leftarrow$ **function** calcNesting$(v, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$
  if $(v.type = L)$ **then**
    assignNestedRectangle$(v, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
    return $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}, v.nestedR]$;
  else if $(u.type = S)$ **then**
    $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}, nr] \leftarrow$ calcNesting$(v$.getLeftChild$(), \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
    assignNestedRectangle$(v, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
    $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}] \leftarrow$ updateSums$(v, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
    return calcNesting$(v$.getRightChild$(), \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
  else if $(u.type = P)$ **then**
    $x_L \leftarrow 0; \quad x_R \leftarrow +\infty; \quad y_T \leftarrow 0; \quad y_B \leftarrow +\infty;$
    for $child \in v.children()$ **do**
      $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}, nr] \leftarrow$ calcNesting$(child, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
      $[x_L, x_R, y_T, y_B] \leftarrow$ findMaximalRect$(v, child, nr, x_L, x_R, y_T, y_B)$;
    $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}] \leftarrow$ resetSums(v, x_L, x_R, y_T, y_B);
    return $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}, new\ Rectangle(x_L, y_T, x_R - x_L, y_B - y_T)]$;

function assignNestedRectangle$(v, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$;
  $x \leftarrow v.r.x; \quad y \leftarrow v.r.y; \quad w \leftarrow v.r.w; \quad h \leftarrow v.r.h;$
  $v.nestedR.x \leftarrow x + d_x \cdot (\Sigma_{1x} - x \cdot \Sigma_{2x}); \quad v.nestedR.w \leftarrow w - d_x \cdot w \cdot \Sigma_{2x};$
  $v.nestedR.y \leftarrow y + d_y \cdot (\Sigma_{1y} - y \cdot \Sigma_{2y}); \quad v.nestedR.h \leftarrow h - d_y \cdot h \cdot \Sigma_{2y};$

$[x'_L, x'_R, y'_T, y'_B] \leftarrow$ **function** findMaximalRect$(v, child, nr, x_L, x_R, y_T, y_B)$
  if $child.r.x = r.x$ **then** $x'_L \leftarrow \max(x_L, nr.x)$;
  if $child.r.y = r.y$ **then** $y'_T \leftarrow \max(y_T, nr.y)$;
  if $child.r.x + child.r.w = r.x + r.w$ **then** $x'_R \leftarrow \min(x_R, nr.x + nr.w)$;
  if $child.r.y + child.r.h = r.y + r.h$ **then** $y'_B \leftarrow \min(y_B, nr.y + nr.h)$;
  return $[x'_L, x'_R, y'_T, y'_B]$;

$[\Sigma'_{1x}, \Sigma'_{2x}, \Sigma'_{1y}, \Sigma'_{2y}] \leftarrow$ **function** updateSums$(v, \Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y})$
  $a_x \leftarrow v.r.x + v.r.w/2; \quad a_y \leftarrow v.r.y + v.r.h/2;$
  $\Sigma_{1x} \leftarrow \Sigma_{1x} + a_x/v.r.w; \quad \Sigma_{2x} \leftarrow \Sigma_{2x} + 1/v.r.w;$
  $\Sigma_{1y} \leftarrow \Sigma_{1y} + a_y/v.r.h; \quad \Sigma_{2y} \leftarrow \Sigma_{2y} + 1/v.r.h;$
  return $[\Sigma'_{1x}, \Sigma'_{2x}, \Sigma'_{1y}, \Sigma'_{2y}]$

$[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}] \leftarrow$ **function** resetSums(v, x_L, x_R, y_T, y_B)
  $\Sigma_{2x} \leftarrow (x_L - x_R + v.r.w)/(2 \cdot d_x \cdot v.r.w); \quad \Sigma_{1x} \leftarrow v.r.x \cdot \Sigma_{2x} + (x_L - v.r.x)/d_x;$
  $\Sigma_{2y} \leftarrow (y_T - x_B + v.r.h)/(2 \cdot d_y \cdot v.r.h); \quad \Sigma_{1y} \leftarrow v.r.y \cdot \Sigma_{2y} + (y_T - v.r.y)/d_y;$
  return $[\Sigma_{1x}, \Sigma_{2x}, \Sigma_{1y}, \Sigma_{2y}]$;

5.3 DAGmap View: A Tool for Space Filling Visualization of DAGs

In this section we describe a program written in Java that visualizes DAGs using space filling techniques. The program, which is called DAGmap view [206], draws specific classes of DAGs, such as trees, planar st-graphs, Two Terminal Series Parallel digraphs (TTSP) and a subset of Minimum Vertex Series Parallel digraphs (MVSP) using space filling techniques. Additionally, it implements three vertex duplication heuristics: a) all vertices having in-degree greater than one are duplicated, b) a recently discovered heuristic where the initial st-graph is transformed into a DAG that admits a DAGmap, and c) it allows the user to specify a set of vertices to be duplicated. DAGmap View can also be used to visualize the whole Gene Ontology (GO). Tools that visualize the whole GO already exist and some notable examples are described in Baehrecke et al. [23] and Symeonidis et al. [194]. Both of these tools convert the GO DAG into a tree before the visualization. They actually visualize all paths from the source vertex (e.g., the "all" term) to every other vertex. The tool of Baehrecke et al. [23] facilitate queries of DNA microarray data in the context of Gene Ontology categories, while the tool of Symeonidis et al. [194] allows the simultaneous visualization of the miRNA regulatory network and the GO categories of the targeted genes. Other tools like GOfish [32], Amigo [79], CGAP [56] and OBO-Edit [80], visualize the GO with expanding lists or trees, which can manipulate small hierarchies, but face difficulties for large hierarchies.

DAGmap View has been developed for testing new DAGmap algorithms. For this reason the current version of the program lacks connections with microarray data as the program of Baehrecke et al. [23] does. However our program incorporates novel ideas on data visualization such as separation of the layout and hierarchy presentation functions. This allows navigation without changing the mental map of the user. Additionally our program implements the DAGmap algorithms that have been developed during this thesis. Another new feature is that the user can see both the whole GO as a DAGmap as well as small parts of the hierarchy in node-link representation. Last but not least our program was built with the excellent graph library of Dimitris Andreou [15] and for this reason it either equals or outperforms the other competitive programs in speed, stability and extensibility. For example, new heuristic DAGmap algorithms can easily be incorporated into the program. In the next version of the program, we plan to implement features such as queries and connections with other biological data so that to facilitate its use and make it appealing to biologists.

5.4 Description

DAGmap View is written in Java for portability. The main form of the program can be seen in Fig. 5.7 and Fig. 5.12. From the control panel, the user can select the treemap layout algorithm, the number of visible layers, the border size, and he/she can zoom in and out the image and move the image up, down, left and right.

In DAGmap View, four treemap layout algorithms are implemented; namely the slice and dice, the stip, the squarified and the ordered layouts. Initially the code of these layouts was based on the code that is publicly available on the

web page[1] on the history of treemaps which is maintained by Ben Shneiderman. Later in order to improve the performance of the program we programmed the squarified layout algorithm in a more efficient way (see Appendix II).

The program can read Gene Ontology OBO v1.2 format files and simple interaction format files (.sif) (see the Cytoscape manual[2] for the format of .sif files). It exports a DAGmap to a variety of vector graphics formats as well as image formats using the VectorGraphics package of FreeHEP library [3]. The vector graphics formats include PostScript, PDF, EMF, SVF, and Flash SWF. The image formats include GIF, PNG, JPG and PPM. Additionally, DAGmap View uses the FlexiGraph graph library which is maintained by Dimitris Andreou [15]. FlexiGraph is a general framework for graphs. It offers various advantages such as a neat API, node and edge instances can be shared among many graphs, thorough graph event support, a nice node/edge decoration framework, many graph algorithms and input-output routines. In the next version of DAGmap View we plan to take advantage of the new input-output functions of FlexiGraph and allow the program to read and write Graph Modeling Language (.gml) files. Finally the program has an option to show small subgraphs of GO in node-link representation using hierarchical layout. The node-link visualization is accomplished with the JGraph library[4] while the layout algorithm is from the JGraph Layout engine library.

The user selects a rectangle R_u by pressing the left mouse button over it. The color of rectangle R_u changes to denote that rectangle R_u and the corresponding vertex u are selected. Additionally the color of the rectangles of all duplicates of u changes. The default selection color is green and the default color of the duplicates of the selected vertex is a darker shade of green (see Fig. 5.7). However these default colors can change from the options window (see Fig. 5.8). From the options window the user can also change the default colors of relationships "is_a", "part_of", "regulates", "positively_regulates" and "negatively_regulates", the lines' color and width, the text that appears on the rectangles as well as the font and color of the text.

The user may see the attributes of the selected vertex, e.g., description, node ID and name, by right clicking on the rectangle of the vertex and then selecting from the popup menu the "display vertex attributes" choice. Then a new window appears that displays the attributes of the selected vertex (see Fig. 5.9).

He/she can also choose to display (in node-link representation) the subgraph of the original DAG (before any vertex duplication) which is induced from the vertices that belong to all paths that start from the source and terminate at the selected vertex. This functionality of the program resembles the graphical presentation of search results, using AmiGO, in the official GO web page [5]. For example when the user selects the rectangle that corresponds to vertex "regulation of RNA metabolic process" (GO:0051252) of Fig. 5.7 the subgraph that consists of all paths from the root to vertex GO:0051252 is shown in a new window (see Fig. 5.10). Note that in gene ontology the arrows point from a term to a more general term. Namely they point towards the source vertices.

[1] http://www.cs.umd.edu/hcil/treemap-history/
[2] http://www.cytoscape.org/
[3] http://java.freehep.org/vectorgraphics/
[4] http://www.jgraph.com/
[5] http://www.geneontology.org/

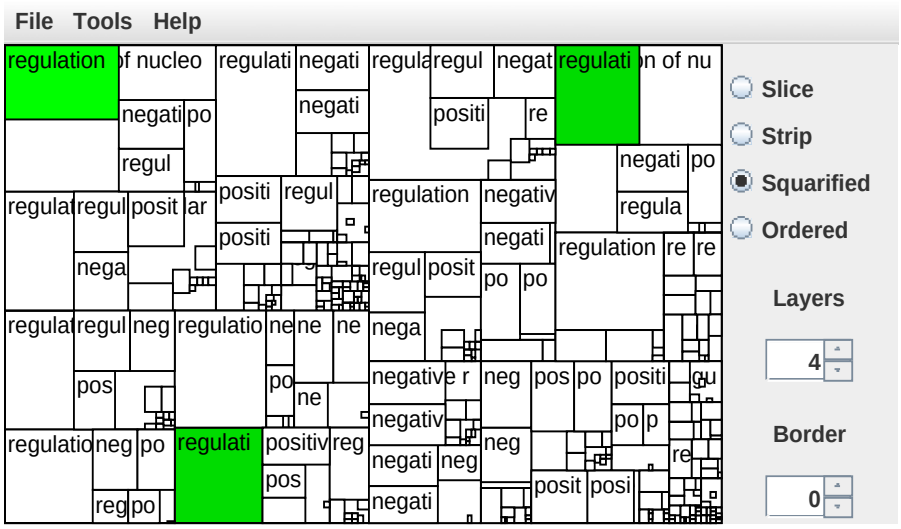


Figure 5.7: DAGmap View demo: The GO is converted into a tree and visualized as treemap. The subtree rooted at "regulation of metabolic process" is shown. The rectangle that correspond to term "regulation of RNA metabolic process" is selected. The selected term and its duplicates are painted green. The color of duplicates is slightly darker than that of the selected term.

The tool has implemented many novel ideas such as space filling drawing of vertices and of edges of a DAG, separate layout and hierarchy presentation functions, zooming without changing the size of the nesting borders and keeping the relative position of the rectangles constant during zooming and navigation. Figure 5.11 shows that although the nesting borders change on-line the relative positions of the rectangles are not affected. This is a consequence of the separation of the layout and nesting functions. Navigation through the hierarchy is another example where this separation improves the stability of the DAGmap and helps preserving the mental map of the user. When a user double clicks on the rectangle of "cellular metabolic process" of Fig. 5.12 the subtree rooted into this vertex is displayed (see Fig. 5.13). The aspect ratios of the rectangles in this figure are different from those in Fig. 5.12 since the rectangle of node "cellular metabolic process" has been stretched to fit into the display rectangle. However the relative position of the rectangles remains the same. The user may return to the previous display by clicking the right mouse button and then by choosing "go back" in the popup menu. Figure 5.14 shows how the functions zoom out and move left work. The zoom out, zoom in, move up, move down, move left, move right functions resemble the google earth magnification/navigation strategy. The difference is that in our program the nesting borders do not change while the user magnifies or shrinks a DAGmap. The hierarchy presentation function illustrates the structure of a hierarchy using a number of techniques including the cushion [213], the nested [183], and the cascaded [127] presentations. In DAGmap view we implemented nested presentation and we plan to implement cascaded and cushion presentations in the next release of the pro-

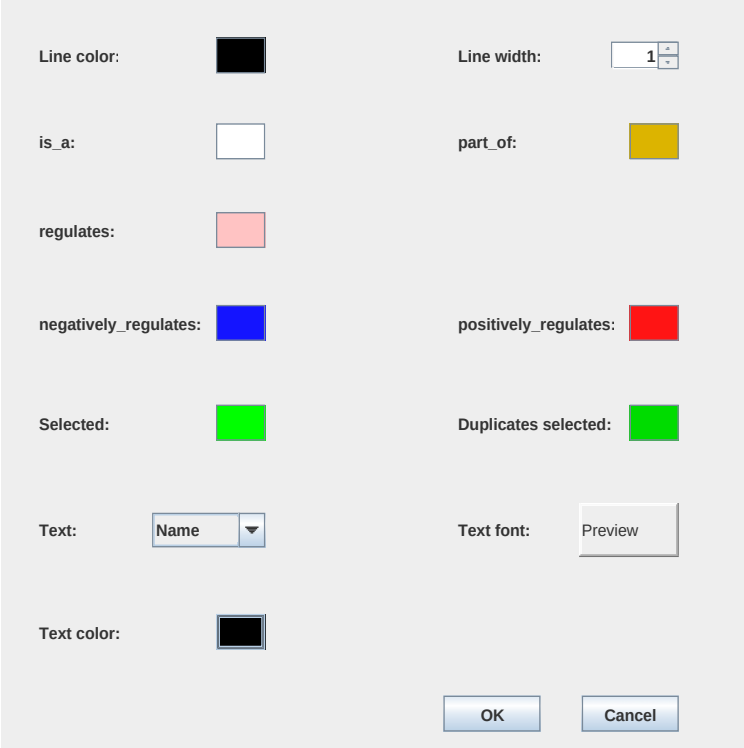


Figure 5.8: DAGmap View demo: The options window.

ID : GO:0051252
Name : regulation of RNA metabolic process
Definition : "Any process that modulates the frequency, rate or extent of the chemical reactions and pathways involving RNA." [GOC:ai]
Synonyms :
"regulation of RNA metabolism" EXACT

Figure 5.9: DAGmap View demo: Some attributes of selected term “regulation of RNA metabolic process”.

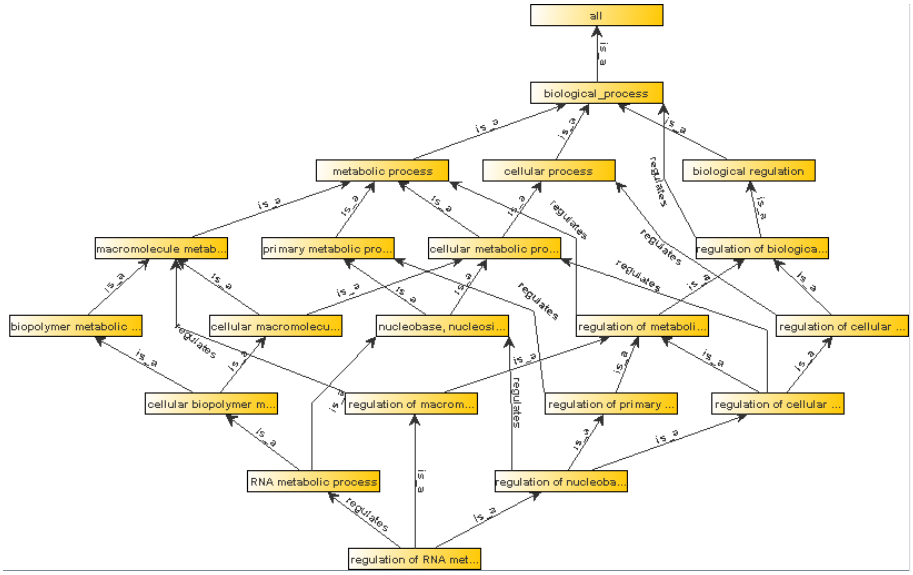


Figure 5.10: DAGmap View demo.

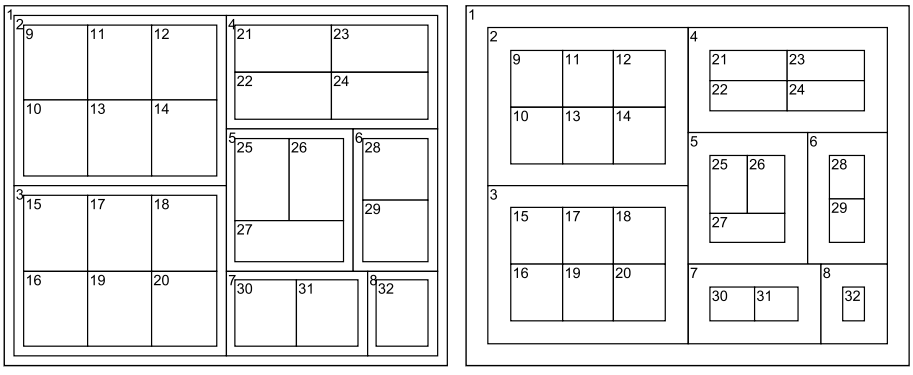


Figure 5.11: DAGmap View demo: The user changes the nesting borders online.

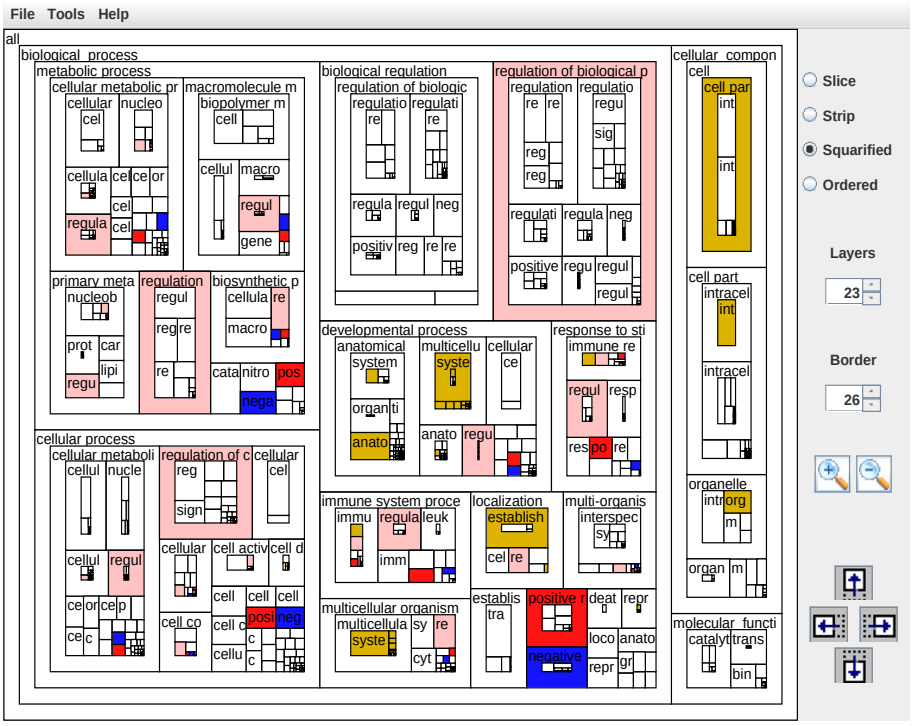


Figure 5.12: DAGmap View demo: The GO is converted into a tree and visualized as a treemap.

gram. In Sect. 5.2 we describe the theory of how nesting works in trees and TTSP digraphs.

5.5 Recognized Classes of DAGs

DAGmap View recognizes and displays trees, TTSP digraphs and MVSP digraphs. A DAG is tree if and only if it is connected and has $n-1$ edges. It is rooted tree if it has a single root and all vertices are reachable from the root. The recognition algorithm for TTSP digraphs is described in [208]. The drawing algorithm for TTSP digraphs is described in Sect. 2.5.1. To test whether or not a DAG G is MVSP digraph the program computes the inverse line digraph $L^{-1}(G)$ and then tests if $L^{-1}(G)$ is TTSP digraph. If $L^{-1}(G)$ is TTSP digraph then a DAGmap drawing of it where only the edges appear may correspond to a DAGmap drawing of G where only the vertex rectangles are shown. We recall here that a DAGmap of $L^{-1}(G)$ may not correspond to a DAGmap of G (see Sect. 2.5.2). In a postprocessing step the program decides whether or not a DAGmap of $L^{-1}(G)$ is a DAGmap of G. The most challenging algorithms to implement are the one-dimensional DAGmap recognition and drawing algorithms (see Algorithm 9) since they require planarity testing and a planar embedding of a single source DAG G respectively. These two algorithms will be available in the next version of the program.

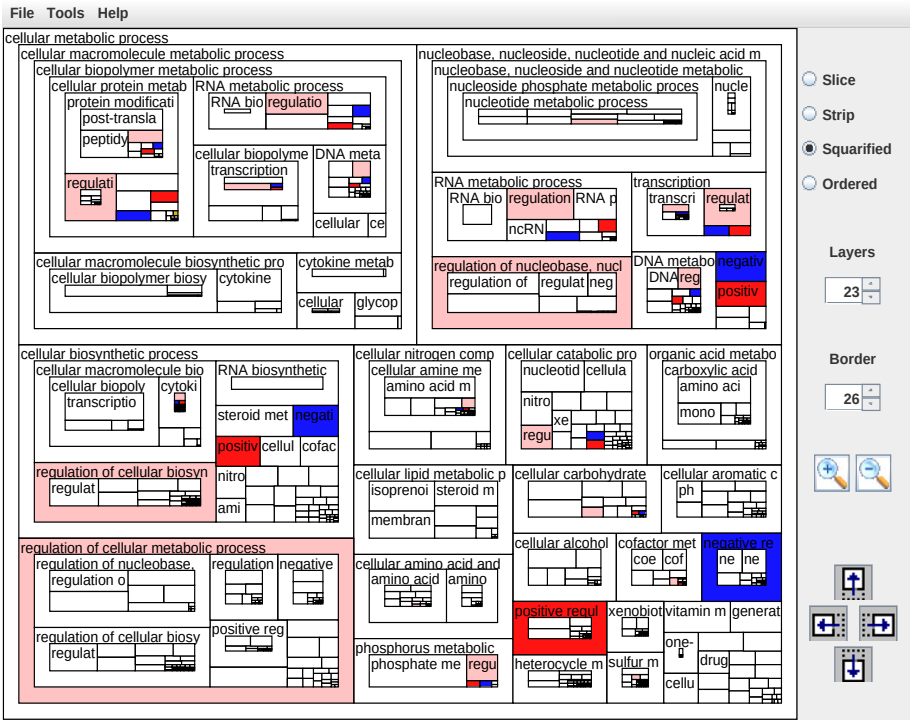


Figure 5.13: DAGmap View navigation demo: When a user double clicks on the border of "cellular metabolic process" of Fig. 5.12 the subtree rooted into this node is displayed.

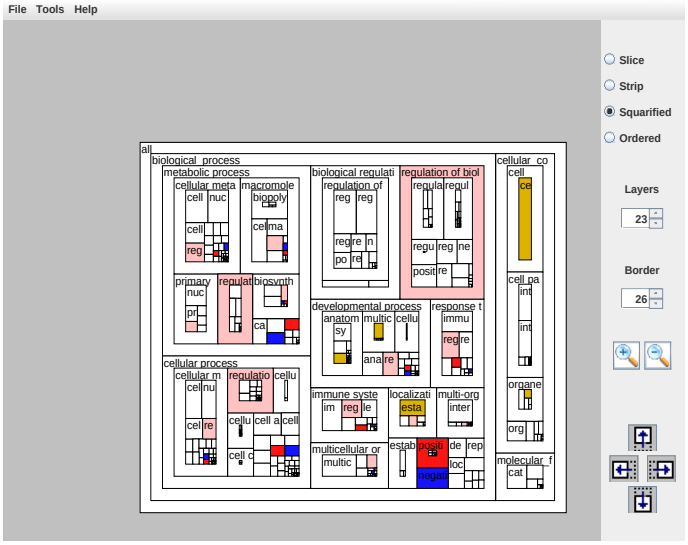


Figure 5.14: DAGmap View demo: Zoom out followed by move left and move down of the image.

5.6 Duplication Heuristics

DAGmap View implements three duplication heuristics. The first heuristic is that a DAG is converted into a tree by duplicating all vertices that have in-degree greater than one. The second heuristic is that an st-graph is converted into a DAG that admits a DAGmap (see Chap. 4). The third heuristic is that the user can specify groups of vertices that can be drawn without duplications. Figure 5.15 shows the patterns of vertices that are currently handled by the program and two examples from GO where these patterns are drawn first without duplications and next with duplications.

5.7 Discussion

Real DAGs, such as GO, differ significantly from both trees and TTSP digraphs. Symeonidis et al. [194] propose to convert a DAG into a tree by duplicating all vertices that have in-degree greater than one. In this thesis we propose to convert an st-graph G into a DAG that admits a DAGmap by a clever duplication strategy that takes into account the dominators and the post-dominators of G. One question is how our vertex duplication algorithm performs on GO. If we apply this algorithm to a copy of GO from year 2005 the results are very encouraging. At that time GO contained about 20000 terms and its structure was close to the structure of a tree. Symeonidis et al. [194] reported that converting that GO DAG into a tree produces a tree with 100000 nodes (400% increase). On the other hand converting that GO DAG into a TTSP digraph produces a DAG with 45000 vertices (125% increase) which is a significant reduction on the number of duplications.

During the last four years the complexity of GO increased significantly. Nowadays GO contains 28000 terms and converting it into a tree produces a tree with 1000000 nodes (3600% increase). Unfortunately the current copy of GO has few pairs of immediate dominator - immediate post-dominator and the resulting DAG, if we apply our vertex duplication algorithm, has 920000 vertices (3300% increase) which is not a significant reduction on the number of duplications.

To estimate how much we can reduce the number of vertex duplications using graph decomposition techniques, we decompose GO into its connected, biconnected [201] and triconnected [94, 86, 62] components. After this decomposition we end up with series, parallel and rigid graph components. Note that this decomposition is a finer decomposition than the one that we propose using dominators and post-dominator to separate subgraphs of DAG G. Assuming that only vertices that belong to rigid components are duplicated, this decomposition has the potential to produce fewer vertex duplications than our heuristic decomposition algorithm and provides a theoretical lower bound on the number of vertex duplication. However deriving practical heuristic duplication algorithms based on graph decomposition into connected, biconnected and triconnected components is not feasible due to complicated algorithms that are involved in the decomposition and due to the fact that some components need to be reassembled before we apply a duplication heuristic. A version of "biological process" from June 2009 had 16657 vertices. This DAG is decomposed into 18 non-trivial biconnected components. The largest biconnected component

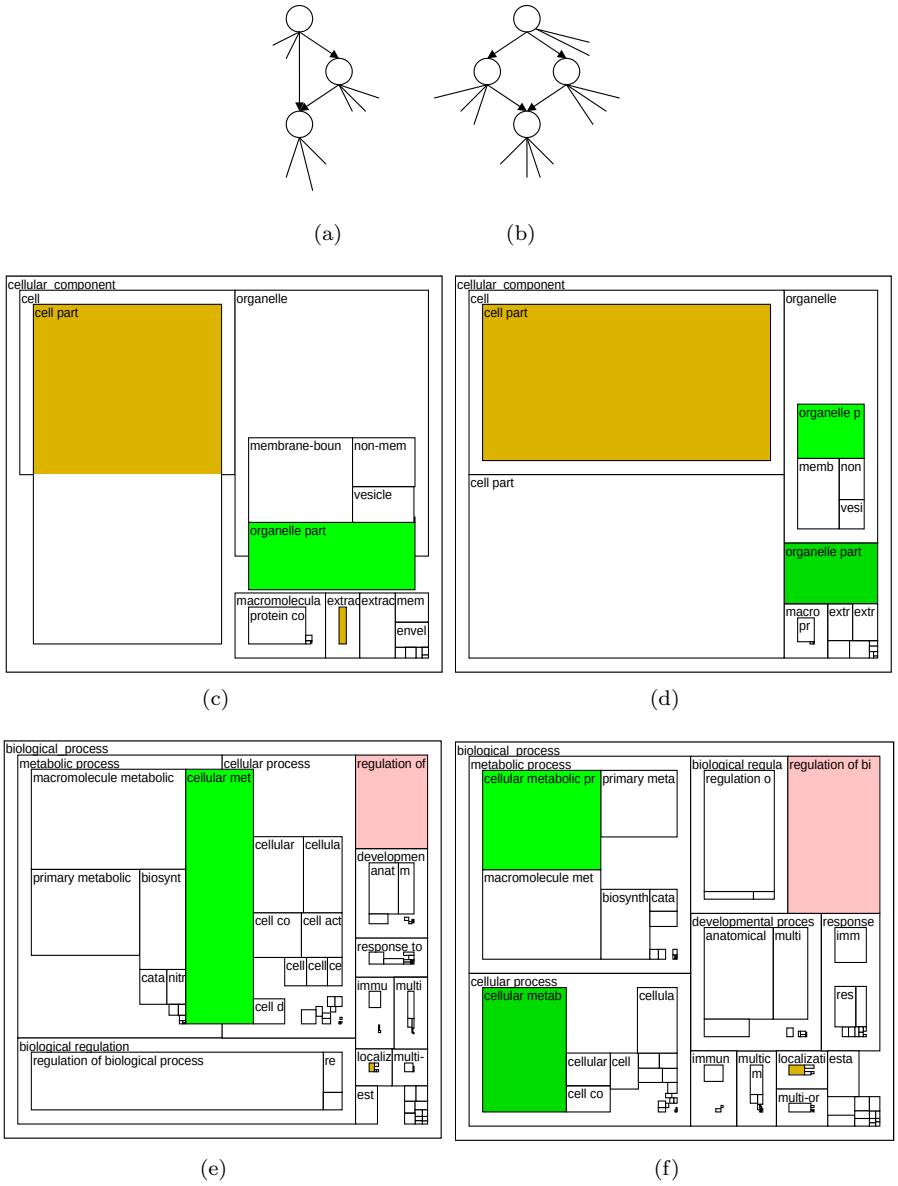


Figure 5.15: (a) and (b) Patterns of vertices that DAGmap View can draw without duplications. (c) The three terms: cellular component, cell and cell part as well as the three terms: cellular component, organelle and organelle part form patterns similar to the one of Fig. (a). These terms are drawn without duplication. Compare with Fig. (d) where the same terms are drawn with duplications. With green color are displayed the selected terms and their duplicates. (e) and (f) Drawing without and with duplications of the four terms: biological process, metabolic process, cellular process and cellular metabolic process. These terms form a pattern similar to the one of Fig. (b).

has 14372 vertices and is further decomposed into 18 triconnected components. One rigid component is huge and consists of 10144 vertices and 26709 edges. Duplicating the nodes of this rigid component produces a DAG with almost half a million nodes. This is a clear indication that graph decomposition techniques in GO are not going to provide a way to significantly reduce the vertex duplications.

However a preliminary analysis shows that if the user specifies small subgraphs of G that can be drawn without duplications then we could achieve significant reduction in the number of duplicated vertices. A preliminary analysis shows that a reduction of up to 50% is possible (see Fig. 5.15). Note that these small subgraphs may be part of a triconnected component.

Chapter 6

Brain Functional Connectivity Estimation from EEG

A recent trend in brain functional connectivity analysis is to model the interdependencies between brain signals with networks [52, 191]. The interdependence between brain areas is estimated using multivariate neurophysiological signals (EEG, MEG, ECoG) and/or haemodynamic response images (fMRI). Then a network is formed by corresponding either brain areas or channels to vertices and by considering an edge between two vertices if and only if the estimated interdependence is above a threshold. The next step in the analysis is to measure some networks statistics and characterize the network. Then using the network characterization one can draw conclusions on the effect of illnesses or of cognitive loads on functional connectivity [189, 168, 153, 132] (see Fig. 6.1 for an overview of the analysis).

Motivated by the importance that the networks play in brain functional connectivity analysis we created a program called BrainNetVis which visualizes functional networks and computes a number of networks statistics. BrainNetVis is described in Chap. 8. A design choice in the construction of the program was to model functional networks with valued networks instead of dichotomous networks. A unified description of valued networks, especially those that have values in the interval $(0, 1]$, exists in Chap. 7. However before describing the program and the theory behind it we should explain how we construct a functional network from EEG data. In this chapter we briefly introduce EEG and then we describe three synchronization measures: a nonlinear interdependence measure, coherence and partial directed coherence. These three measures have been chosen after an extensive study in linear and nonlinear synchronization measures [170]. The advantage of coherence is that it is well known and widely accepted. The advantage of the nonlinear synchronization measure is that it is quite robust to noise. Finally the advantages of partial directed coherence is that it can measure causality, it is a multivariate measure and its matlab code is publicly available.

In this chapter the common spatial patterns technique is also described. This spatial filtering technique improves the signal to noise ratio in multivariate data

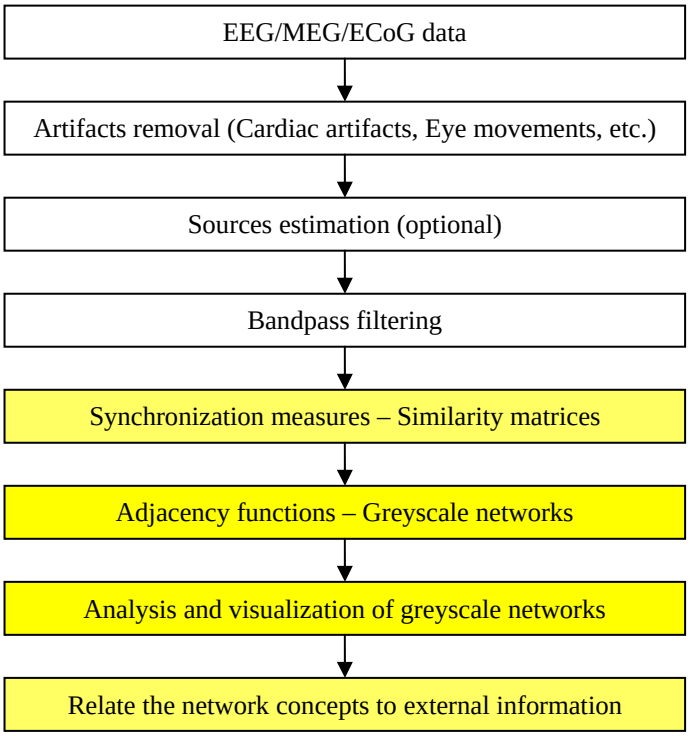


Figure 6.1: An overview of neurophysiological data processing and how network concepts can be used to extract information from data.

measured from experiments that belong to two different classes. We use this technique in Chap. 8 to enhance the calculation of power spectral density.

Finally we briefly discuss a few methods of how to choose a threshold. This description serves as a link between previous studies that were based on binary networks and our study that focuses on valued networks. Additionally choosing a threshold is needed in the case studies of Chap. 8 where we calculate statistics of both binary and greyscale networks in order to compare the results.

6.1 EEG origin

The recording of the electrical activity of the brain [i.e., the electroencephalogram (EEG)], either the ongoing activity or the changes of activity related to a given sensory or motor event [the event-related potentials (ERPs)], provides the possibility of studying brain functions with a high time resolution but with a relatively modest spatial resolution. Scalp EEG measures the summed activity of post-synaptic currents. An action potential in a pre-synaptic axon causes the release of a neurotransmitter into the synapse that diffuses across the synaptic cleft and binds to receptors in a post-synaptic dendrite, resulting in a flow of ions into or out of the dendrite, which in turn results in compensatory currents in the extracellular space. It is these extracellular currents that generate EEG

voltages. Although post-synaptic potentials generate the EEG signal, it is not possible for a scalp EEG to determine the activity within a single dendrite or neuron. Rather, a surface EEG reading is the summation of the synchronous activity of 10-100 millions of neurons that have similar spatial orientation (radially to the scalp).

6.2 EEG Oscillatory Patterns

The theory of the whole-brain-work explains brain function in cognitive processing on the basis of oscillatory activity [140, 26]. The very existence of a theory on the oscillations shows that EEG oscillatory or rhythmic activity, be it spontaneous or evoked, needs special emphasis. The EEG rhythmic activity is divided into *windows* or *bands* by frequency.

Delta is the frequency range up to 3 Hz. It tends to be the highest in amplitude and the slowest waves. It is seen normally in adults in slow wave sleep.

Theta is the frequency range from 4 Hz to 7 Hz. This range has been associated with reports of relaxed, meditative, and creative states.

Alpha is the frequency range from 8 Hz to 12 Hz. Hans Berger named the first rhythmic EEG activity he saw, the *alpha wave*. This is activity in the 8-12 Hz range seen in the posterior regions of the head on both sides, being higher in amplitude on the dominant side. It is brought out by closing the eyes and by relaxation. It was noted to attenuate with eye opening or mental exertion. This activity is now referred to as *posterior basic rhythm*. In addition to the posterior basic rhythm, there are two other normal alpha rhythms that are typically discussed: the *mu rhythm* and a temporal *third rhythm*.

Mu rhythm is alpha-range activity that is seen over the sensorimotor cortex. It characteristically attenuates with movement of the contralateral arm (or mental imagery of movement of the contralateral arm).

Beta is the frequency range from 12 Hz to about 30 Hz. It is seen usually on both sides in symmetrical distribution and is most evident frontally. Low amplitude beta with multiple and varying frequencies is often associated with active, busy or anxious thinking and active concentration. Rhythmic beta with a dominant set of frequencies is associated with various pathologies and drug effects. It may be absent or reduced in areas of cortical damage. It is the dominant rhythm in patients who are alert or anxious or who have their eyes open.

Gamma is the frequency range approximately $26-100$ Hz. Gamma rhythms are thought to represent binding of different populations of neurons together into a network for the purpose of carrying out a certain cognitive or motor function.

6.3 EEG Limitations and Artifacts

EEG has several limitations. Most important is its poor spatial resolution. Due to volume conduction (electrical spread), which is defined as the transmission of electric or magnetic fields from an electric primary current source through biological tissue towards measurement sensors, the activity of a single source is measurable in many channels. Especially close-by electrodes are highly syn-

chronized which reflects redundancies in the measurement rather than brain interaction. Differentiating true brain dynamics from artifactual results caused by volume conduction is a highly non-trivial task.

In EEG recordings, electric potential is measured on the scalp surface, and used to make inferences about brain activity. Although potentials relative to infinity are often considered in theoretical derivations, in practice only potential differences can be measured. Thus EEG measurements always involve the potential difference between two sites. This is accomplished using differential amplifiers, which include the measure electrodes, a reference electrode, and an isolated common electrode that takes the place of true ground. It is often assumed that the reference electrode is an inactive reference, such that the potential at that site is constant across time. However, the reference sites such as the earlobes, mastoids, or noise, have been shown to vary in their electrical potentials during a measurement epoch. The active reference electrode continues to confound many EEG studies.

Apart from the volume conduction and the reference issues, EEG is mixed with environmental and hardware system noise 6.2(a) and with biological artifacts. Biological artifacts often contaminate EEG records and generally pose a more serious problem than environmental or system noise. Common artifact sources include whole body movement, heart, muscle, eyes, and tongue. EEG records containing large artifacts are often discarded. Artifact removal by computer is typically successful only for the largest artifacts. Such automated artifact editing is severely limited because the frequency bands of biological artifacts substantially overlap the important EEG bands, making distinctions between artifact and brain signal difficult. Independent Component Analysis (ICS) is a method that has been successfully applied to EEG artifact removal [90] (see Fig. 6.2(b)).

6.4 Common Spatial Patterns

A particularly popular and powerful signal processing technique for improving the signal to noise ratio in EEG-based motor imagery classification tasks is called common spatial patterns (CSP) [114, 210] which is a technique to analyze multichannel data based on recordings from two classes (conditions). Let $\Sigma^{(L)} \in \mathbb{R}^{n\times n}$ and $\Sigma^{(F)} \in \mathbb{R}^{n\times n}$ be the estimates of the covariance matrices of the band-pass filtered EEG signal in the two conditions (e.g., left hand and foot imagination):

$$\Sigma^{(c)} = \frac{1}{|\mathcal{I}_c|} \sum_{i\in\mathcal{I}_c} X_i X_i^T \ , \quad c \in \{L, F\}$$

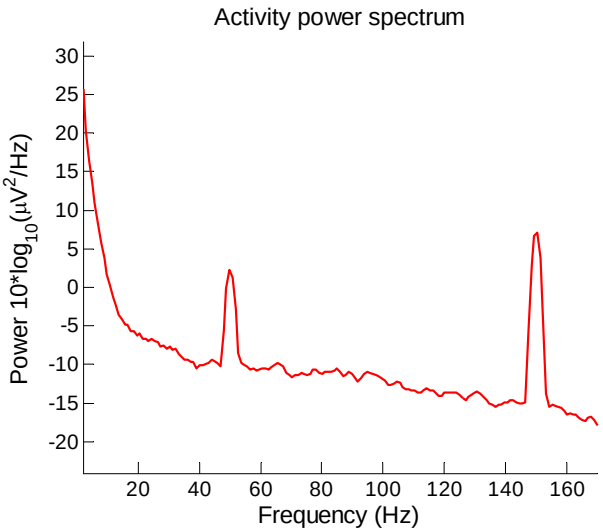
where $\mathcal{I}_c$ is the set of indices corresponding to trials belonging to class $c \in \{L, F\}$ and $|\mathcal{I}_c|$ denotes the size of set $\mathcal{I}_c$.

By extracting the eigenvectors and eigenvalues from Σ

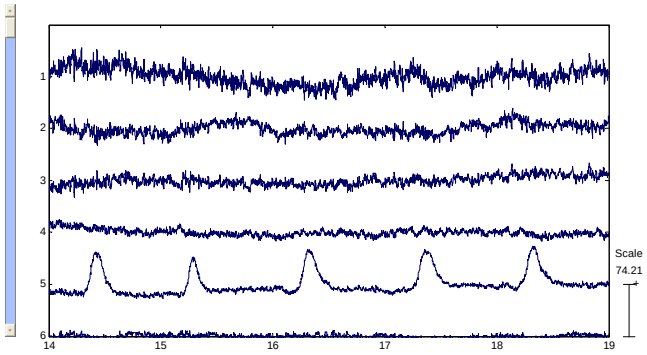
$$\Sigma = \Sigma^L + \Sigma^F = U_0 \Lambda U_0^T$$

we can calculate the spatial factors matrix Q and the whitening matrix P.

$$Q = U_0 \Lambda^{1/2} \quad \text{and} \quad P = \Lambda^{-1/2} U_0^T$$



(a) AC Line noise at 50 Hz and its odd harmonic at 150 Hz



(b) Eye blinks at ICA component 5

Figure 6.2: Examples of EEG artifacts. a) Alternating Current (AC) line noise shown by taking the power spectrum of a channel. The AC line operates at 50 Hz. b) Applying ICA to BCI data. We see that component 5 corresponds to eye blinks.

Then we calculate matrices $S^{(L)} = P\Sigma^{(L)}P^T$ and $S^{(F)} = P\Sigma^{(F)}P^T$ which have common eigenvectors and the corresponding eigenvalues sum to one (i.e., $\Lambda^{(L)} + \Lambda^{(F)} = I$ and $\Lambda^{(L)}_{ii} \leq \Lambda^{(L)}_{jj}$ when $i < j$).

$$S^{(L)} = U\Lambda^{(L)}U^T \quad \text{and} \quad S^{(F)} = U\Lambda^{(F)}U^T$$

Taking the first r (resp. last r) (e.g. r = 3) eigenvectors from U, we obtain $U_r \in \mathbb{R}^{r\times n}$ to compute the spatially filtered EEG signal

$$X^{(F)}_r = QU_rU_r^TPX \quad (\text{resp. } X^{(L)}_r = QU_rU_r^TPX)$$

6.5 Synchronization Measures

6.5.1 Preliminaries

A manifold is a topological space in which every point has a neighborhood which resembles Euclidean space, but in which the global structure may be more complicated. Given two manifolds M and N, a bijective map f from M to N is called a *diffeomorphism* if both $f : M \to N$ and its inverse $f^{-1} : N \to M$ are differentiable. Let M be a compact manifold called the phase (or state) space. A *dynamical system* on M is a diffeomorphism $\varphi : M \to M$ (discrete time) or a vector field X on M (continuous time). In both cases the time evolution corresponding with an initial position $x_0 \in M$ is denoted by $\varphi_t(x_0)$: in the case of discrete time $t = n \in \mathbb{N}$ and $\varphi_n = (\varphi)^n$ (function composition); in the case of continuous time $t \in \mathbb{R}$ and $t \to \varphi_t(x_0)$ is the X integral curve through x_0. A stochastic dynamical system is a dynamical system subjected to the effects of noise. An observable is a smooth function y that to each point of phase space M associates a number ($y : M \to \mathbb{R}$).

6.5.2 Synchronization of Weakly Coupled Systems

Synchronization is a process wherein many systems adjust a given property of their motion due to a suitable coupling configuration, or to an external forcing. Synchronization is a fundamental nonlinear phenomenon, and it plays an important role in various fields of science and engineering [158]. Very often it is found in living systems, being observed on a level of single cells, physiological subsystems, organisms and even on the level of populations [192]. Sometimes, this phenomenon is essential for a normal functioning of a system, e.g. for a coordinated motion of several limbs or for the performance of a pacemaker, where many cells fire synchronously, and in this way produce a macroscopic rhythm that governs respiration, heart contraction, etc. Additionally, there is evidence that suggests that populations of neurons can code information through the synchronization of their responses and that long-range oscillatory synchronization implements coordinated communication between various areas in the brain [46]. Sometimes, the onset of synchrony leads to a severe pathology, e.g. when abnormal synchronization of a group of neurons can result in the development and propagation of an epileptic seizure and in case of the Parkinson's disease, when locking of many neurons results in the tremor activity [179].

Synchronization between dynamical systems has been an active research topic since the time of Huygens (1665) [31]. The research in synchronization

phenomena started with the study of periodic oscillators, while recently the attention has moved to chaotic oscillators. When oscillators are coupled, many different synchronization phenomena take place: complete synchronization [157], phase synchronization [166], lag synchronization [167], generalized synchronization [169, 160], and noise-induced synchronization [129], to mention only a few.

During the last decade there has been research interest in the comprehension of synchronization phenomena when oscillating elements are constrained to interact in a complex network topology. The analysis of synchronization processes has benefited from the advance in the understanding of the topology of complex networks, but it has also contributed to the understanding of general emergent properties of networked systems.

6.5.3 Nonlinear Measures

Phase-space Reconstruction

Packard et al. [154] addressed the problem of how to reconstruct a phase-space picture of a finite dimensional dynamical system using a series of observed values sampled at regular intervals from the system. They proposed a heuristic reconstruction using as phase-space coordinates a vector of time-delayed values of the observed quantity.

Takens [197] proved that, if for some dynamical system with time evolution φ_t, we know the functions $t \to y(\varphi_t(x))$, $x \in M$, then we can obtain information about the original dynamical system (and manifold) using the map $\Phi_{(\varphi,y)} : M \to \mathbb{R}^{2d+1}$, defined by

$$\Phi_{(\varphi,y)}(x) = [y(x),\ y(\varphi_1(x)),\ \ldots,\ y(\varphi_{2d}(x))]^T$$

This map is an embedding of M onto its image $\Phi_{(\varphi,y)}(M) \subset \mathbb{R}^{2d+1}$, that is a smooth one-to-one map which has a smooth inverse.

Takens also considered the case that the values of the observable y of a dynamical system is only determined for a discrete set $\{0, \tau, 2\tau, 3\tau, \ldots\}$ of value of t; $\tau > 0$ and he proved that, under generic assumptions, the topology of, and dynamics in the limit set

$$L^+(x) = \{x' \in M \mid \exists n \to \infty \text{ with } \varphi_{n\tau}(x) = x'\}$$

of the point $x \in M$ are determined by the time series $\{y(\varphi_{n\tau}(x))\}_{n=0}^{\infty} = \{y(n)\}_{n=0}^{\infty}$ of values of the observable. More precisely, the positive limit set $L^+(x)$ is "diffeomorphic" with the set of limit points of the following sequences in $\mathbb{R}^{2d+1}$

$$\{[y(\varphi_{n\tau}(x)),\ y(\varphi_{(n+1)\tau}(x)),\ \ldots,\ y(\varphi_{(n+2d)\tau}(x))]^T\}_{n=0}^{\infty}$$

The meaning of "diffeomorphic" is that there is a smooth embedding of M into $\mathbb{R}^{2d+1}$ mapping $L^+(x)$ bijectively to this set of limit points.

Taken's theorems [197] and their sequels [174] are existence proofs. They do not directly show how to get a suitable time delay τ or embedding dimension $m = 2d + 1$ from a finite time series. Empirical and heuristic criteria are employed for selecting τ and m. Usually, a choice of τ is the value for which the autocorrelation function first passes through zero, while m is determined using variations of false nearest neighbor statistics [57].

Another practical issue is the effect of filtering on the time series before the reconstruction. Badii et al. [22] have shown that certain filters, such as linear low-pass filters that are described by differential equations, add to the dynamics of the experimental system, distorting the reconstruction of the attractor. Mitschke [137] claimed that filters that do not introduce phase shifts do not significantly affect the reconstruction and he proposed the use of acausal filters.

A Robust Interdependence Measure

Arnhold et al. [18] presented a measure for characterizing statistical relationships between two time series. This measure, which is based on the theory of recurrence of dynamical system, has been used for EEG studies by several researchers [17, 161]. Suppose two scalar time series $\{y(n)\}$ and $\{z(n)\}$ with $n = 1, \ldots, N$ have been measured from dynamical systems Y and Z, respectively. At first, the dynamics of the systems are reconstructed using delay coordinates

$$\mathbf{y}(n) = [y(n), y(n+\tau), \ldots, y(n+(m-1)\tau)]^T$$
$$\mathbf{z}(n) = [z(n), z(n+\tau), \ldots, z(n+(m-1)\tau)]^T$$

with an embedding dimension m and a delay time τ for $n = 1, \ldots, N-(m-1)\tau$.

Let $r_{n,j}$ and $s_{n,j}$, $j = 1, \ldots, k$, denote the time indices of the k nearest Euclidean neighbors of $\mathbf{y}(n)$ and $\mathbf{z}(n)$, respectively. Temporally correlated neighbors are excluded by means of a Theiler [202] correction: $|r_{n,j} - n| > m \cdot \tau$ and $|s_{n,j} - n| > m \cdot \tau$. For each $\mathbf{z}(n)$, the mean squared Euclidean distance to its k nearest neighbors is given by

$$R_n^{(k)}(Z) = \frac{1}{k} \sum_{j=1}^{k} |\mathbf{z}(n) - \mathbf{z}(s_{n,j})|^2$$

and the Y-conditioned mean squared Euclidean distance is defined by replacing $s_{n,j}$ with $r_{n,j}$ (see Fig. 6.3)

$$R_n^{(k)}(Z/Y) = \frac{1}{k} \sum_{j=1}^{k} |\mathbf{z}(n) - \mathbf{z}(r_{n,j})|^2$$

The average square distance of $\mathbf{z}(n)$ to all remaining points in $\{\mathbf{z}(n)\}$ is given by

$$R_n(Z) = \frac{1}{N'-1} \sum_{j=1, j\neq n}^{N'} |\mathbf{z}(n) - \mathbf{z}(j)|^2$$

where $N' = N - (m-1)\tau$

If the dynamics of Z is independent of Y, then there is no particular relation between $r_{n,j}$ and $s_{n,j}$ and

$$R_n(Z) \approx R_n^{(k)}(Z/Y) \gg R_n^{(k)}(Z)$$

holds. In contrast, if closeness in Y implies closeness in Z, then it follows that

$$R_n(Z) \gg R_n^{(k)}(Z/Y) \approx R_n^{(k)}(Z)$$

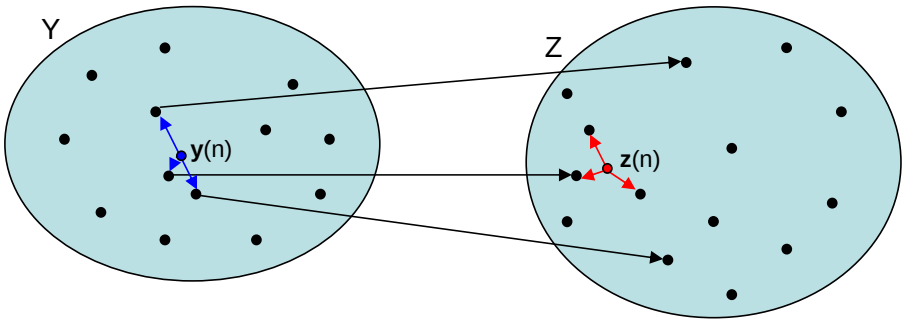


Figure 6.3: An example of how $R_n^{(k)}(Z)$ and $R_n^{(k)}(Z/Y)$ are calculated. The phase spaces of the two dynamical systems are denoted by Y and Z. The points represent m-dimensional vectors. The parameter k is fixed to 3. The calculation of $R_n^{(k)}(Z)$ is based on the three nearest neighbors of point $\mathbf{z}(n)$ in phase space Z which are shown by red arrows. The calculation of $R_n^{(k)}(Z/Y)$ is based on the three points in phase space Z that correspond to the three nearest neighbors of $\mathbf{y}(n)$ in phase space Y which are shown by blue arrows. For simplicity the Theiler correction is not considered.

Based on these considerations, Arnhold et al. [18] defined the interdependence measure:

$$S(Z/Y) = \frac{1}{N'} \sum_{n=1}^{N'} \frac{R_n^{(k)}(Z)}{R_n^{(k)}(Z/Y)} \tag{6.1}$$

while Quian Quiroga et al. [161] defined the interdependence measure:

$$N(Z/Y) = \frac{1}{N'} \sum_{n=1}^{N'} \frac{R_n(Z) - R_n^{(k)}(Z/Y)}{R_n(Z)} \tag{6.2}$$

which is more robust than $S(Z/Y)$ since $R_n(Z)$ is less dependent on structure and dimensionality in Z than $R_n^{(k)}(Z)$.

The measures $S(Y/Z)$ and $N(Y/Z)$ are defined in complete analogy to (6.1) and (6.2) in order to test whether Y is dependent on Z. Although, in general, $S(Y/Z) \neq S(Z/Y)$ (resp. $N(Y/Z) \neq N(Z/Y)$) these measures are not suitable to infer driver response relationships since the asymmetry may reflect the different degrees of freedom of the two systems.

6.5.4 Linear Measures

The linear measures are suitable to describe the linear relations between signals. Although the neurophysiological signals are generated by systems of inherently nonlinear nature (neuronal populations) there is evidence that they can be successfully treated with linear approach and there is no need to apply a nonlinear procedure [35]. Even during epilepsy, when nonlinear effects are most pronounced, we may still expect correct answers[190, 133]. Linear methods are quite simple to implement, have fewer restrictions about the input

data characteristics and can quickly give an insight into the nature of the problem. Additionally, when recorded signals are intermixed from many different neuronal populations, the data effectively have a form of colored noise. In such cases, there is a high risk that nonlinear methods may perform worse than linear methods.

Spectral Estimation

Let $\mathbf{X}$ denote a stationary discrete-time m-dimensional stochastic process. A value of the process at time $t \in \mathbb{T}$ can be expressed as:

$$\mathbf{X}(t) = [X_1(t), X_2(t), \ldots, X_m(t)]^T$$

where $\mathbb{T}$ is a set of consecutive integers. Without loss of generality we assume that every process X_i has zero mean. A realization of process $\mathbf{X}$ is a sequence (of vectors) $\mathbf{x}(t)$, $t \in \mathbb{T}$. A multiple time series is regarded as such a realization. The frequency representation, $\widetilde{\mathbf{X}}(f)$, of process $\mathbf{X}$ is a complex valued function describing amplitudes and phases of frequency components of process $\mathbf{X}$ at frequency f. The Power Spectral Density (PSD) matrix of signal $\mathbf{X}$ is defined as

$$\mathbf{S}(f) = \widetilde{\mathbf{X}}(f)\widetilde{\mathbf{X}}^H(f) = \begin{pmatrix} S_{11}(f) & \ldots & S_{1m}(f) \\ \vdots & \ddots & \vdots \\ S_{m1}(f) & \ldots & S_{mm}(f) \end{pmatrix} \tag{6.3}$$

where the superscript H represents transposition and complex conjugate. The PSD matrix consists of autospectral terms $S_{ii}(f)$, for $i = 1, \ldots, m$, and cross spectral terms $S_{ij}(f)$, for $i \neq j$, from all possible pairings.

In order to analyze data in frequency domain the PSD matrix $\mathbf{S}(f)$ has to be estimated. Popular methods are the periodogram and its variations. These methods use the Fast Fourier Transform (FFT) algorithm which evaluates the spectral power of a signal in a fast and effective way. Multivariate autoregressive PSD estimation is another approach that performs better than periodogram when applied to short data segments. In this approach a model is fitted to the data resulting in a set of model parameters. The whole analysis is then conducted on the model parameters, not on the data samples.

Assume that the process $\mathbf{X}(t)$ can be described by the following pth-order autoregressive equation (MAR(p))

$$\mathbf{X}(t) = \sum_{r=1}^{p} \mathbf{A}(r)\mathbf{X}(t-r) + \mathbf{E}(t) \tag{6.4}$$

where $\mathbf{A}(r)$ are $m \times m$ coefficient matrices and $\mathbf{E}(t) = [E_1(t), E_2(t), \ldots, E_m(t)]^T$ ia a zero mean uncorrelated noise vector with covariance matrix $\mathbf{\Sigma}$.

In frequency domain (6.4) can be written as:

$$\mathbf{X}(f) = \mathbf{H}(f)\mathbf{E}(f) \tag{6.5}$$

where

$$\mathbf{H}(f) = \bar{\mathbf{A}}^{-1}(f) = (\mathbf{I} - \mathbf{A}(f))^{-1} \quad \text{and} \quad \mathbf{A}(f) = \sum_{r=1}^{p} \mathbf{A}(r)e^{-jr2\pi f}$$

From (6.5) we obtain the spectral matrix

$$\mathbf{S}(f) = \mathbf{H}(f)\mathbf{\Sigma}\mathbf{H}^H(f) \tag{6.6}$$

where $\mathbf{\Sigma}$ is the noise covariance matrix.

Spectral Estimation of Non-Stationary Signals

When estimating spectral properties of signals in the frequency domain, the data record length determines the statistical quality of the estimate. The model should be fitted to stationary records of data where the statistical properties of the signals do not change in time. Typically, the assumption of stationarity is not valid for longer data records. When the record length is too short, the estimates become unreliable. Although linear modeling allows to perform spectral estimations even for very short data segments, nevertheless the limitation for a minimal record length is a serious problem in analysis of dynamical processes. Data recordings of rapidly changing phenomena, like evoked potentials, cannot be split into long and stationary epochs.

One solution to the above problem is to allow the model parameters to change in time. The class of adaptive methods is based on this idea. Another approach is an application of the short sliding window method to parametric spectral estimation. The procedure was proposed in [63] and applied in [106]. It can be used when multiple repetitions of a process are available for recording. In that case, all the repetitions will be treated as different realizations of the same stochastic process. This assumption allows for utilization of all realizations in the estimation of spectral quantities. In this way even data records of short length can still produce reliable spectral estimates. The whole nonstationary data record can be divided into short time windows, often short enough to treat the data inside them as stationary. If the window size is w, we must estimate pm^2 parameters from wmN_R data points, where N_R is the number of repetitions. The bigger N_R is, the shorter the window length w can be chosen. The actual decision about the window size must consider a balance between the time resolution of the results and the quality of the fit.

During model fitting the data covariance matrix is calculated for each repetition of the experiment. Then the averaged covariance matrix is calculated over all repetitions and next, this average is used to determine the parameters of a model within a given window.

Coherence and Partial Coherence

Coherency is defined by means of elements of the spectral matrix $\mathbf{S}$ and depends on signals X_i and X_j and on frequency f.

$$C(X_i, X_j, f) = C_{ij}(f) = \frac{S_{ij}(f)}{\sqrt{S_{ii}(f)S_{jj}(f)}} \tag{6.7}$$

Its modulus squared, the coherence $|C_{ij}(f)|^2$ (or magnitude square coherence), is a function describing the linear time invariant dependence of two processes at frequency f and it takes values in the $[0, 1]$ range (with 0 indicating independence and 1 indicating maximal interdependence (synchronization) of the two processes at frequency f).

In neuroscience coherence is usually computed using the Welch's method as follows:
Given two simultaneously measured discrete time series $\{x(n)\}_{n\in\mathbb{T}}$ and $\{y(n)\}_{n\in\mathbb{T}}$, $\mathbb{T} = \{1 \dots N\}$ the cross spectral density function S_{xy} is derived via the FFT of the cross-correlation. The individual autospectral density functions S_{xx} and S_{yy} are also derived via the FFT transform. Then we compute

$$\gamma_{xy}(f) = \frac{|\langle S_{xy}(f)\rangle|^2}{|\langle S_{xx}(f)\rangle||\langle S_{yy}(f)\rangle|} \tag{6.8}$$

Where $\langle . \rangle$ indicates window averaging.

Nolte et al. [148] proposed to use the imaginary part of coherency which is insensitive to false connectivity arising from volume conduction. The imaginary part of coherency is only sensitive to synchronization of two processes which are time-lagged to each other and insensitive to zero-lagged volume conduction effects. However it is likely that this approach misses parts and in the worst case all of the brain interaction.

A measure of conditional dependence of processes X_i and X_j, given some other processes $Y \subset \{X_1, \dots, X_m\} \setminus \{X_i, X_j\}$, is provided by the partial coherency.

$$C_{ij|Y}(f) = \frac{S_{ij|Y}(f)}{\sqrt{S_{ii|Y}(f)S_{jj|Y}(f)}} \tag{6.9}$$

Brillinger [47] proved that

$$S_{ij|Y} = S_{ij}(f) - \mathbf{S}_{iY}(f)\mathbf{S}_{YY}(f)^{-1}\mathbf{S}_{Yj}(f)$$

where $\mathbf{S}_{iY}(f) = \widetilde{X}_i(f)\widetilde{\mathbf{Y}}(f)^H$, $\mathbf{S}_{YY}(f) = \widetilde{\mathbf{Y}}(f)\widetilde{\mathbf{Y}}(f)^H$, and $\mathbf{S}_{Yi}(f) = \widetilde{\mathbf{Y}}(f)\widetilde{X}_i(f)^*$
When $\mathbf{Y} = \mathbf{X} \setminus \{X_i, X_j\}$, then we have

$$C_{ij|Y}(f) = \frac{\mathbf{M}_{ij}(f)}{\sqrt{\mathbf{M}_{ii}(f)\mathbf{M}_{jj}(f)}} \tag{6.10}$$

where $\mathbf{M}_{ij}$ is a minor (determinant) of $\mathbf{S}_{XX}$ with i-th row and j-th column removed. Partial coherence is the modulus squared $|C_{ij|Y}(f)|^2$ of partial coherency and takes values within the $[0, 1]$ range similar to ordinary coherence, but it is nonzero only when the relation between channel i and j is direct.

When a fitted model in (6.4) is available, a simpler and more convenient means of computing (6.10) is possible. For each frequency f, consider the partition of

$$\bar{A}(f) = I - A(f) = [\bar{a}_1(f), \bar{a}_2(f), \dots, \bar{a}_m(f)]$$

into columns. Then the partial coherence between X_i and X_j at each frequency is given by:

$$|C_{ij|Y}|^2 = \frac{| < \bar{a}_i(f), \bar{a}_j(f) > |^2}{< \bar{a}_i(f), \bar{a}_i(f) >< \bar{a}_j(f), \bar{a}_j(f) >} \tag{6.11}$$

where $< \bar{a}_i(f), \bar{a}_j(f) >= \bar{a}_i^H(f)\Sigma^{-1}\bar{a}_j(f)$ and Σ is the prediction error covariance matrix.

Partial Directed Coherence and Granger Causality

Coherence and partial coherence are symmetric measures and do not show the direction of information flow between two stochastic processes. Baccalá et al. [20] introduced Partial Directed Coherence (PDC), $\pi_{i \leftarrow j}(f)$, as a frequency domain description of the directed linear influence of process X_j on process X_i at frequency f when the influence due to the remaining processes is discounted. Assuming that process $\mathbf{X}$ is modeled as an autoregressive process (6.4), PDC, based on the factorization of partial coherence (6.11), is defined as:

$$\pi_{i \leftarrow j}(f) = \frac{|\bar{A}_{ij}(f)|}{\sqrt{\sum_{l=1}^{m} |\bar{A}_{lj}(f)|^2}} \tag{6.12}$$

where

$$\bar{A}_{ij}(f) = \delta_{ij} - \sum_{r=1}^{p} a_{ij}(r) e^{-\mathbf{j} r 2\pi f}$$

for $f \in [-0.5, 0.5]$, $\mathbf{j} = \sqrt{-1}$, and $\delta_{ij} = 1$ whenever $i = j$ and $\delta_{ij} = 0$ otherwise. Alternatively, one might also consider $|\bar{A}_{ij}(f)|$ as a non-normalized version of PDC.

The autoregressive representation of $\mathbf{X}$ is closely related to the concept of (linear) Granger causality, which is a fundamental tool to describe the causal relationship between time series. According to the original definition of Granger [84], one time series $\mathbf{X}$ causes another series $\mathbf{Y}$, if the onestep ahead prediction of $\mathbf{Y}$ based on the past of $\mathbf{Y}$ and that of any relevant auxiliary variables $\mathbf{Z}$ can be improved (in the mean square sense) by adding the past of $\mathbf{X}$ to the set of predictor variables. In the context of multivariate time series $\mathbf{X}$ with autoregressive representation (6.4), this leads to the following equivalent definition: one component X_j linearly Granger-causes another component X_i if the coefficients $a_{ij}(r)$ do not vanish uniformly for all lags $r \in \{1, \ldots, p\}$ [97, 65, 128]. Thus, linear Granger causality describes the direct linear effect of one component X_j on another component X_i.

It follows from (6.12) that PDC $\pi_{i \leftarrow j}(f)$ vanishes for all frequencies $f \in [-0.5, 0.5]$ if and only if in (6.4) it is $A_{ij}(r) = 0$ for all lags r. Consequently, the PDC can be viewed as a frequency domain measure of Granger causality.

Recently, Shelter et al. [176] and Takahashi et al. [195, 196] investigated the statistical adequacy of PDC. Shelter et al. [176] studied the significance level for null hypothesis testing of zero connectivity.

$$H_0 : \ \pi_{i \leftarrow j}(f) = 0$$

Takahashi et al. [195] studied the related problem of defining asymptotic confidence interval for PDC when H_0 does not hold and they obtained rigorous results by assuming that the data generating processes are MAR of finite known order. In a subsequent paper Takahashi et al. [196] extended these results to encompass the infinite order MAR processes case under mild conditions that guarantee PDC convergence by fitting MAR models whose order depend on the number of available time samples.

The normalization of PDC 6.12 by the factor $\sqrt{\sum_{l=1}^{m} |\bar{A}_{lj}(f)|^2}$ has been criticized that it induces several pitfalls associated with time series scaling and

with its interpretability. In order to avoid numerical problems associated with time series scaling Baccalá et al. [21] defined the *new partial directed coherence*. On the other hand, Schelter [177], in order to avoid more general interpretability problems including the problems associated with time series scaling, introduced the *renormalized partial directed coherence* and calculated confidence intervals and significance levels.

PDC Implementation issues

A MATLAB program that calculates PDC and its statistics for a given data set is publicly available on PDC homepage[1]. In the offered package, there are four algorithms that estimate a MAR model from the time series data. The Nutall-Strand, the mlsm, the Vieira Morf, and the QR artfit algorithms. The code of Nutall-Strand and Vieira Morf algorithms is similar to the code that exists in the book of Marple [130]. Schlögl [178] carried out a comparison study of the Nutall-Strand, the Vieira Morf, and the QR artfit algorithms and he recommends the Nutall-Strand algorithm. Apart from the PDC the package includes code to calculate the partial coherence and the directed transfer function [105, 106, 68].

6.6 Thresholding

The application of a dependency measure to a set of n time series produces an $n \times n$ matrix S. The entries of S belong to interval $[0, 1]$ and are estimations of the true dependencies among the time series. In order to improve the signal to noise ratio we apply a, so called, adjacency function to the entries of S. An adjacency function is a monotonically increasing function that maps the interval $[0, 1]$ into $[0, 1]$. The most widely used adjacency function is the threshold function, $thres$, which implements hard thresholding involving the threshold parameter θ. Specifically,

$$thres_\theta(x) = \begin{cases} 1 & \text{if } s_{ij} \geq \theta \\ 0 & \text{if } s_{ij} < \theta \end{cases}$$

The threshold function converts pairwise dependencies into an adjacency matrix and leads to a binary or dichotomous network. Hard thresholding using the threshold function leads to intuitive network concepts but it may lead to a loss of information. To avoid the disadvantages of hard thresholding, soft thresholding functions have been proposed, such as the power function with parameter β

$$power_\beta(x) = x^\beta$$

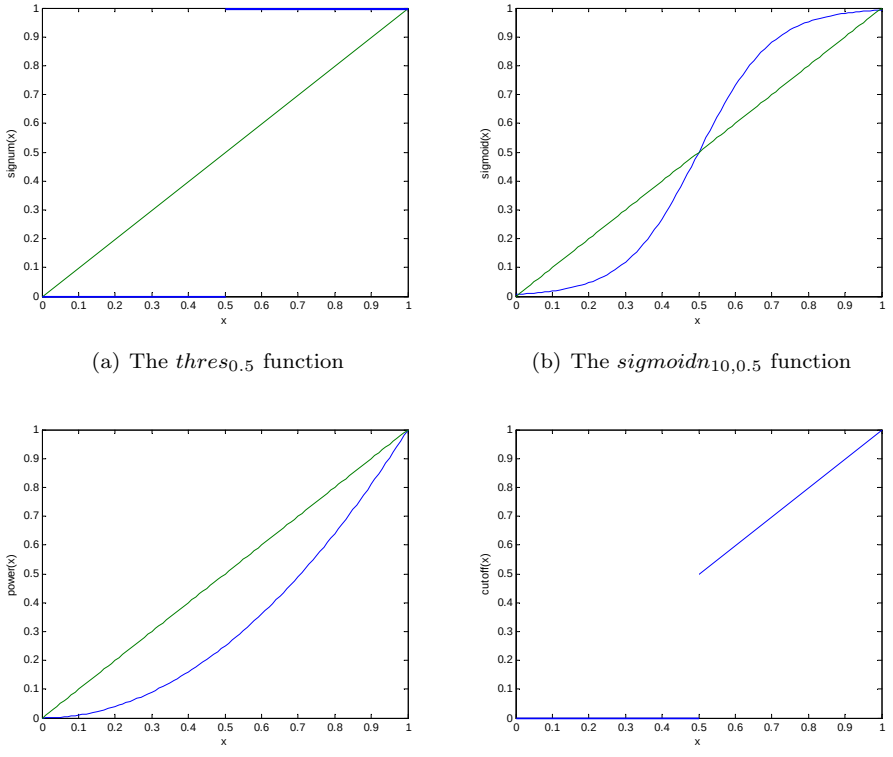
and the sigmoid function with parameters α and θ

$$sigmoid_{\alpha,\theta}(x) = \frac{1}{1 + e^{-\alpha(x-\theta)}}$$

The sigmoid function maps 0 to $\frac{1}{1+e^{\alpha\theta}}$ and 1 to $\frac{1}{1+e^{\alpha(\theta-1)}}$. These values are very undesirable since we want 0 to be mapped to 0 and 1 to 1. To remedy this problem we propose the normalization

$$sigmoidn_{\alpha,\theta}(x) = \frac{sigmoid_{\alpha,\theta}(x) - sigmoid_{\alpha,\theta}(0)}{sigmoid_{\alpha,\theta}(1) - sigmoid_{\alpha,\theta}(0)}$$

[1] http://www.lcs.poli.usp.br/∼baccala/pdc/



(a) The $thres_{0.5}$ function

(b) The $sigmoidn_{10,0.5}$ function

(c) The $power_2$ function

(d) The $cutoff_{0.5}$ function

Figure 6.4: Example of threshold functions

A combination of hard and soft thresholding is also possible. For example, small values which are bellow a threshold θ are attributed to noise and are set to zero.

$$cutoff_\theta(x) = \begin{cases} x & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

6.6.1 Choosing a threshold value θ

The estimation of the threshold parameter θ of functions $thres$, $sigmoid$ and $cutoff$ is tricky. And even the existence of a single global threshold θ is questioned. We will describe two cases where each dependency link s_{ij} has its own threshold θ_{ij} which is estimated by statistical techniques and some ideas on how to choose a global threshold θ using network concepts.

Some dependency estimation methods, such as the PDC, return a value greater than zero if the dependency between two time series is statistically significant, otherwise they return zero. These methods actually apply a $cutoff$ adjaceny function to each dependency link s_{ij} with different threshold θ_{ij}. For other dependency measures, such as the robust interdependence measure 6.5.3, there is no theory that defines which values are statistically significant. In this

case, one may use statistical hypothesis testing and randomization techniques to estimate a threshold above which the values of a dependency measure are statistically significant [117, 59]. Statistical hypothesis testing essentially consists of a null hypothesis H0 and a discriminating statistic. In our case, the discriminating statistic is the value of a dependency measure, s_{ij}, when applied to time series $\{x_i(n)\}$ and $\{x_j(n)\}$. The null hypothesis is:

H0: Two time series are independent and any positive value returned from a dependency measure is attributed to noise.

Applying a statistical hypothesis test to observed data can result in two outcomes: either the null hypothesis is rejected, or it is not. In the former case there is a probability a that the null hypothesis is rejected even though it is true (Type I Error), in the latter case there is a probability b (Type II Error) that we will fail to reject the null when it is in fact false. The probability a is known as the significance level, its complement $(1 - a)$ is the confidence level. The power of a test $(1 - b)$ is the probability the null hypothesis is correctly rejected.

Given a pair of time series $\{x_i(n)\}$ and $\{x_j(n)\}$, we test hypothesis H0 as follows:

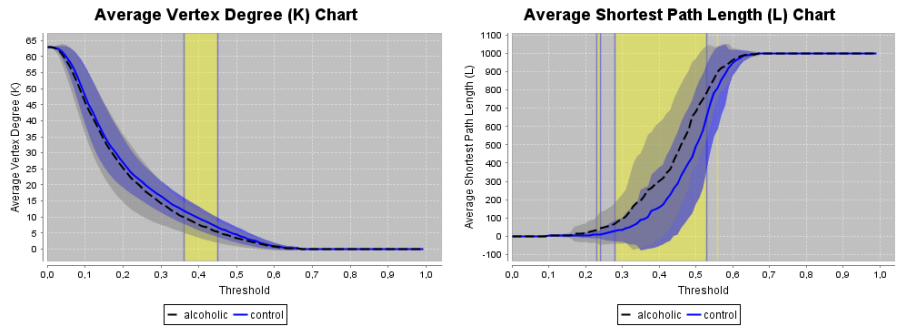
1. Calculate the value s_0 of a dependency measure with input the two time series $\{x_i(n)\}$ and $\{x_j(n)\}$
2. Create 40 pairs of surrogate time series $\{x_{i,k}(n)\}$ and $\{x_{j,k}(n)\}$, $k = 1, \ldots, 40$.
3. For k = 1 to 40
   Calculate the value s_k of the dependency measure with input the time series $\{x_{i,k}(n)\}$ and $\{x_{j,k}(n)\}$
4. Calculate the mean $< s >$ and standard deviation σ_s of s_k, $k = 1, \ldots, 40$
5. Reject hypothesis H0 at significance level $a = 0.05$ if $\frac{<s>-s_0}{\sigma_s} < -1.645$

The surrogate data for hypothesis H0 consist of the one time series ($\{x_i(n)\}$) and permutations of the original data in the other time series ($\{x_j(n)\}$). By randomly shuffling (i.e re-ordering) $\{x_j(n)\}$ any dependency between $\{x_i(n)\}$ and $\{x_j(n)\}$ is destroyed. The new time series $\{x_{j,k}(n)\}$ is a realization of a process that maintains the same amplitude distribution as the original $\{x_j(n)\}$ but without any temporal correlations (an independent and identically distributed noise process). Assuming that the distribution of $\frac{<s>-s_0}{\sigma_s}$ is normal $N(0, 1)$ we design the one-sided test "Reject hypothesis H0 at significance level $a = 0.05$ if $\frac{<s_{ij}>-s_{ij,0}}{\sigma_{s_{ij}}} < -1.645$", and from this test the threshold θ_{ij} is estimated to be $\theta_{ij} =< s_{ij} > +1.645 * \sigma_{s_{ij}}$ [173].

Choosing a threshold value θ using network concepts

Examples on how one can use network concepts to choose a global threshold θ will be presented next.

Oikonomou [149] in his M.Sc. thesis conducted extended experiments with binary networks that encode the synchronization between channel signals during a mental experiment. He worked with data collected from two groups of



(a) Average vertex degree

(b) Average shortest path length

Figure 6.5: Example of how the network statistics K and L differ between the alcoholic (black dashed) and control (blue) patients as the threshold varies. The networks were constructed by estimating the synchronization between the channel signals using the robust interdependence measure proposed by Arnhold et al. [18]

people during working memory experiments. He set up the threshold $\theta = 0, 0.001, 0.002, \ldots, 0.999, 1$ and for a fixed dependency measure and frequency band he created binary networks for each threshold value. Then he calculated for each threshold the mean value and the standard deviation (across networks of the same group) of the average degree K, of the clustering coefficient C and of the average shortest path length L. He found that for some ranges of threshold values the differentiation in K, C and L between the two groups is statistically significant according to t-test with significance level $a = 0.05$, and he proposed to use a threshold in these ranges. Figure 6.5 shows the mean value and standard deviations of K and L of two groups, 40 alcoholic and 40 control patients, during a working memory experiment. One can also see the regions where there are statistically significant differences in the mean values of K and L of the two populations.

Chapter 7

Analysis of Greyscale Networks

Many systems in nature and in technology are modeled by networks where the interaction between a pair of vertices is represented by an edge. We call such networks binary or dichotomous networks because an interaction link either exists or does not exist. A deeper understanding of these systems requires that, in addition to the underlying network structure, information about the strength of interactions is also taken into account. This is accomplished by assigning values to edges, such as the outcome of a synchronization measure when applied to time series data of brain activity (see Chap. 6). A network augmented by edge values is called weighted network in complex networks literature and valued network or valued graph in social networks literature. We will use the term valued network since in graph theory the term "weight" has been associated with distance, cost and dissimilarity whereas term "valued" is more general and does not conflict with the notions of synchronization, dependence and similarity. A *valued network* $N = (V, E, \omega)$ is modeled with a graph $G = (V, E)$ and an edge value function $\omega : E \to \mathbb{R}$ that assigns to each edge $e \in E$ a real value $\omega(e)$.

7.1 Preliminaries: Elements of Algebraic Graph Theory

A square $n \times n$ matrix $W = (w_{ij})$ is called reducible if the indices $1, 2, \ldots, n$ can be divided into two disjoint nonempty sets $i_1, i_2, \ldots, i_\mu$ and $j_1, j_2, \ldots, j_\nu$ (with $\mu + \nu = n$) such that

$$w_{i_\alpha j_\beta} = 0$$

for $\alpha = 1, 2, \ldots, \mu$ and $\beta = 1, 2, \ldots, \nu$.

A matrix is reducible if and only if it can be placed into block upper-triangular form by simultaneous row/column permutations. The adjacency matrix of a digraph is reducible if and only if the digraph is not strongly connected. A square matrix that is not reducible is said to be irreducible.

Theorem 7.1 *[33, 83] Let $G = (V, E)$ be a directed graph with adjacency matrix*

A. The number of walks from vertex v_i to vertex v_j in G with length k is the entry in position (i, j) of the matrix A^k.

Let $W = (w_{ij}) \in \mathbb{C}^{n \times n}$ be an $n \times n$ matrix with complex numbers as entries. A non-zero vector $x \in \mathbb{C}^n$ is an eigenvector of W with corresponding eigenvalue $\lambda \in \mathbb{C}$ if x and λ satisfy the equation

$$Wx = \lambda x$$

The spectrum $\Lambda(A) = \{\lambda_1, \ldots, \lambda_n\}$ of a graph G, with adjacency matrix A, is the multiset of eigenvalues of A, where each eigenvalue appears as many times as is its multiplicity.

The following theorem is valid for symmetric matrices and therefore for undirected graphs.

Theorem 7.2 *[83] Let $W \in \mathbb{R}^{n \times n}$ be real symmetric $n \times n$ matrix, then*

1. *W has real eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n$ and n orthonormal eigenvectors forming a basis of $\mathbb{R}^n$,*
2. *the multiplicity of eigenvalue λ_i is equal to the cardinality of a maximum linear independent set of eigenvectors corresponding to λ_i,*
3. *there is a matrix Q with $Q^T = Q^{-1}$ such that $Q^T W Q = diag(\lambda_1, \ldots, \lambda_n)$.*

For asymmetric nonnegative matrices and directed graphs, we have the following theorem, which is a version of the fundamental Perron-Frobenius Theorem.

Theorem 7.3 *[95, 83] Let $W \in \mathbb{R}^{n \times n}$ be a real nonnegative $n \times n$ matrix, then*

1. *there is a real eigenvalue λ of W such that any other eigenvalue λ' satisfies $|\lambda'| \leq \lambda$. This property may also be stated more concisely by saying that the spectral radius of W is an eigenvalue.*
2. *there is a non-zero eigenvector $\mathbf{x}$ associated with λ having either nonnegative entries or nonpositive entries. If matrix W is irreducible then eigenvector $\mathbf{x}$ has its entries either all positive or all negative.*

7.2 Valued Networks and Real Matrices

A valued network $G = (V, E, \omega)$ corresponds to a real $n \times n$ matrix $W = (w_{ij})$, $i, j \in \{1, 2, \ldots, n\}$, where w_{ij} is equal to value $\omega(e)$ of edge $e = (v_i, v_j)$ if $e \in E$, or to 0 otherwise. Matrix W is symmetric if network G is undirected or asymmetric if G is directed. The mapping from the set of valued networks with n vertices to the set of $n \times n$ real matrices is many-to-one and onto. The mapping is onto since for every $n \times n$ matrix $W = (w_{ij})$, the graph with edge set $E = V \times V$ and edge weights $\omega(v_i, v_j) = w_{ij}$ is mapped onto W. To show that the mapping is many-to-one let G_1 and G_2 be two valued networks that differ only in edge $e = (v_i, v_j)$; network G_1 has edge e with value $\omega(e) = 0$ and network G_2 does not have this edge. Then both G_1 and G_2 are mapped onto the same matrix. To make the mapping one-to-one we can either restrict the edge values or restrict the edge set. In the first case value 0 encodes the absence

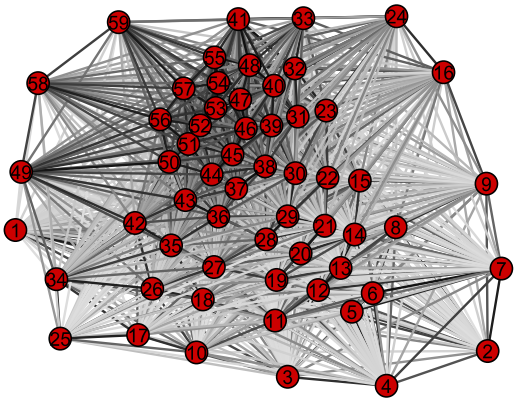


Figure 7.1: In greyscale networks the edge values can be visualized using different shades of grey.

of an edge and is reserved. Therefore the edge values are restricted to $\mathbb{R} \setminus \{0\}$. In the second case the edge set is fixed $E = V \times V$.

Under either of the above restrictions every $n \times n$ matrix $W = (w_{ij})$ defines a valued network. Having restricted the edge values, the matrix rows (or columns) correspond to vertices and there is an edge from vertex v_i to vertex v_j having value w_{ij} if and only if $w_{ij} \neq 0$. Having restricted the edge set to be $E = V \times V$, the edge values $\omega(v_i, v_j)$ correspond to matrix entries w_{ij}. In our study, we assume that value 0 encodes the absence of an edge and that edge values belong to interval $(0, 1]$.

7.3 Greyscale Networks

We consider a subset of valued networks where edge values are restricted to interval $(0, 1]$ and interpreted as strength of dependence or synchronization between vertices. Synchronization of a system with itself makes no sense and we do not allow self-loops. We call these networks *greyscale* since their edge values can be visualized using different shades of grey (see Fig. 7.1). These networks should not be confused with the *grey networks* where the term *grey* means inadequate information [122, 124]. Another candidate name for these networks is *similarity networks*, though the term *similarity* is overloaded.

Definition 7.1 (Greyscale network) *A greyscale network $G = (V, E, \omega)$ consists of a set of vertices V, a set of edges $E \subset V \times V$ and a function $\omega : E \to (0, 1]$ which assigns a real number in $(0, 1]$ to every element $e \in E$. The number $\omega(e)$ is interpreted as the strength of dependence of the system represented by vertex $dest(e)$ on the system represented by vertex $orig(e)$.*

7.3.1 Edge Lengths in Greyscale Networks

In greyscale networks higher edge values indicate stronger dependencies. To define the length of an edge we should at least reverse the order of edge values.

Let $g : (0, 1] \to [0, +\infty)$ be a monotonic non-increasing function, that is if $x_1, x_2 \in [0, 1]$, with $x_1 < x_2$, then $g(x_1) \geq g(x_2)$, then the length, $\ell(e)$, of an edge $e \in E$ is defined to be:

$$\ell(e) = g(\omega(e))$$

Examples of non-increasing functions that have been introduced in complex networks and social networks literature are:

- The linear transformation proposed by Valente and Foreman [203] $g : (0, 1] \to (0, M]$, which is defined by

  $$g(x) = M(1 - x)$$

  where M is a number greater than the diameter of the network.

- The inverse function $g : (0, 1] \to [1, +\infty)$, defined by

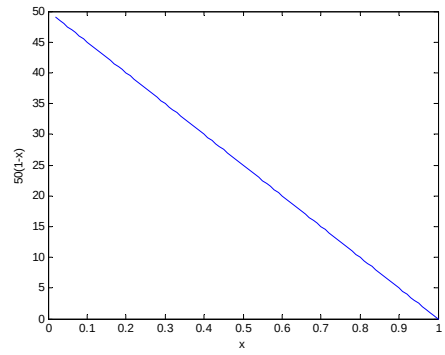
  $$g(x) = \frac{1}{x}$$

- The logarithm function $g : (0, 1] \to [0, +\infty)$, defined by

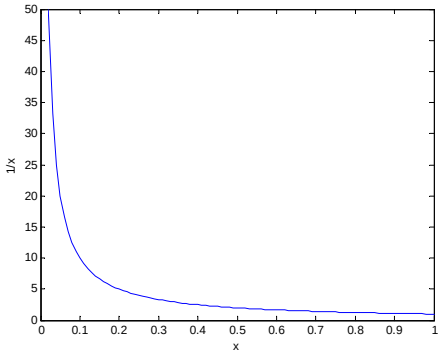
  $$g(x) = -\log_2(x)$$

The linear transformation proposed by Valente and Foreman does not generalize well the graph theoretic notion of distance. It assigns distance 0 to adjacent vertices and distance M to non-adjacent vertices, whereas in graph theory, it is assumed that two vertices have distance 1 when there are adjacent and distance $+\infty$ otherwise. This mismatch in the notion of distances may lead to counter-intuitive results when we run graph theoretic algorithms (see Fig. 7.3).

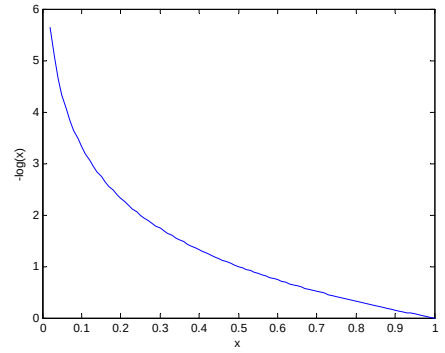
The $\log_2$ function is usually used when edge values are interpreted as probabilities. This function is also related to social and physical laws of information transmission through paths of a network. Katz [109] studied the attenuation of information transmitted between two actors through intermediaries in a social network. He modeled the information transmitted as a base number α, $(0 < \alpha < 1)$, raised to the power of its corresponding path distance ($s_{ij} = \alpha^{d_{ij}}$, where s_{ij} is the information arriving at actor j when one unit of information is transmitted from actor i (and vice versa), and d_{ij} is the path distance of the traveling route). The key point is that information traveling long path distances becomes much weaker than information traveling short ones [54]. By solving equation $s_{ij} = \alpha^{d_{ij}}$ for d_{ij} we get $d_{ij} = \log_\alpha s_{ij}$. We can easily generalize the last equation for greyscale networks. Especially when vertices i and j are adjacent the last equation relates the edge length with the attenuation of information transmitted through the edge ($l_{ij} = \log_\alpha s_{ij}$). It is reasonable to assume that if vertex i transmits one unit of information then at vertex j arrive $\alpha \cdot w_{ij}$ units of information, where w_{ij} is the value of edge (i, j). Then the length of edge is given by $l_{ij} = \log_\alpha(\alpha \cdot w_{ij}) = 1 + \log_\alpha w_{ij}$. If we choose $\alpha = 0.5$ we have $l_{ij} = 1 - \log_2 w_{ij}$. This equation is similar, up to a constant, to function $g(x) = -\log_2(x)$ and superior to function $g(x)$ in the following respect. It is a proper generalization of the graph theoretic distance since it maps edge strength 1 to edge length 1, while $g(x)$ fails to be a proper generalization



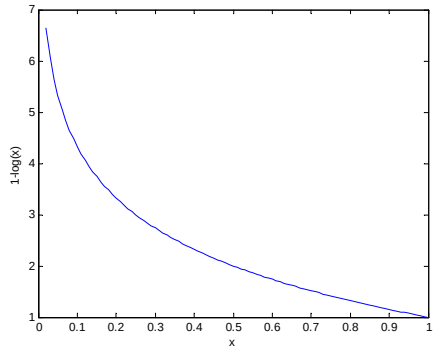
(a) The linear transformation proposed by Valente and Foreman $g(x) = M(1-x)$



(b) The inverse function $g(x) = \frac{1}{x}$

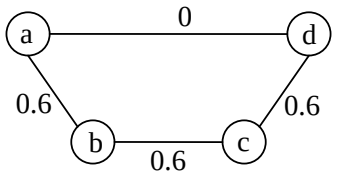


(c) The logarithm function $g(x) = -log_2(x)$

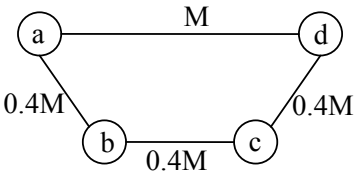


(d) The shifted logarithm function $g(x) = 1 - log_2(x)$

Figure 7.2: Example of non-increasing functions that are used to transform the strength of a dependence into length between pairs of vertices



(a) Edge strength



(b) Edge length

Figure 7.3: An example of counter-intuitive behavior when function $g(x) = M(1-x)$ is used to transform interdependence into distance. The path $a \rightarrow d$ has smaller distance than the path $a \rightarrow b \rightarrow c \rightarrow d$ and shortest-paths algorithms chose it for information transmission. However there is no direct interdependence between vertices a and d and the information between vertices a and d passes through vertices b and c.

since it maps edge strength 1 to edge length 0. Finally, we propose the distance function $g' : (0, 1] \rightarrow [1, +\infty)$, defined by

$$g'(x) = 1 - \log_2(x)$$

As far as we know this function has not been mentioned in the literature.

7.3.2 Distances in Greyscale Networks

Distances Based on Path Lengths

The *length or distance of a path* from vertex u to vertex v is the sum of the lengths of the edges of the path. The shortest path distance from vertex u to vertex v, denoted by $d_G(u, v)$, is the minimum distance across all paths from u to v.

$$d_G(u, v) = \min_{p \in P_{uv}} (\sum_{e \in p} \ell(e)) \tag{7.1}$$

where P_{uv} is the set of paths from u to v. If vertex v is unreachable from vertex u then $d_G(u, v) = +\infty$.

The *importance or the weight of a path* is the product of the values of the edges of the path. Optionally, the importance of paths with many vertices may be attenuated by a decay factor $\alpha \in (0, 1]$.

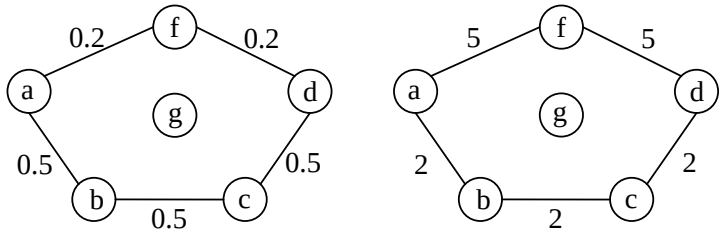
$$\text{imp}(p) = \prod_{e \in p} (\omega(e)\alpha) \tag{7.2}$$

For example, in the network of Fig. 7.4 the distance of path $a - b - c - d$ is 6 and the distance of path $a - f - d$ is 10. The shortest path distance from a to d is $d_G(a, d) = \min(6, 10) = 6$. Note that $d_G(a, g) = +\infty$. The importance of path $a - b - c - d$ is $0.5 \cdot 0.5 \cdot 0.5 = 0.125$ and the importance of path $a - f - d$ is $0.2 \cdot 0.2 = 0.04$. Now assume that every time information is traveling along an edge it is attenuated by a factor $\alpha = 0.1$. Then the importance of path $a - b - c - d$ is $(0.5 \cdot 0.1) \cdot (0.5 \cdot 0.1) \cdot (0.5 \cdot 0.1) = 0.000125$ and the importance of path $a - f - d$ is $(0.2 \cdot 0.1) \cdot (0.2 \cdot 0.1) = 0.0004$.

Distances Based on Structural Equivalence

Using shortest paths is one way to define distance (or dissimilarity) between two vertices. One may also define distance between two vertices according to their neighborhoods. The idea is that vertices with identical neighborhoods play the same role in a network and therefore are considered to be functionally close. In practice statistical and heuristic algorithms are used to measure the similarity and dissimilarity of vertices according to their role in a network. Similarity and dissimilarity (or distance) functions have been introduced in fuzzy sets theory, social networks, biological networks and machine learning literature. Before we give some well known examples we introduce the concept of negation functions that are used to convert similarity into dissimilarity and vice versa.

Negation functions: They are used in fuzzy set theory to convert similarity into dissimilarity. In order of any function $g : [0, 1] \rightarrow [0, 1]$ to be considered a



(a) Edge strength (b) Edge length

Figure 7.4: Edge strength is converted into edge length using the inverse function $g(x) = 1/x$. Then from the edge strengths we calculate the importance and from edge lengths the distance of a path.

fuzzy negation, it must satisfy at least the following two axiomatic requirements [112]:

1. $g(0) = 1$ and $g(1) = 0$ (boundary condition)
2. For all $x, y \in [0, 1]$, if $x < y$, then $g(x) \geq g(y)$, that is, g is monotonic nonincreasing.

A negation function is strict if and only if

3. $g(x)$ is continuous
4. For all $x, y \in [0, 1]$, if $x < y$, then $g(x) > g(y)$

Examples of negation functions are:

- The Sugeno class of functions [193] defined by

$$g_\alpha(x) = \frac{1 - x}{1 + \alpha x} \tag{7.3}$$

  where $\alpha \in (-1, +\infty)$. For each value of the parameter α we obtain one particular negation function.

- The Yager class of functions [217] defined by

$$g_\alpha(x) = (1 - x^\alpha)^{1/\alpha} \tag{7.4}$$

  where $\alpha \in (0, +\infty)$

Euclidean distance: Burt [53] was the first who proposed to use the Euclidean distance to measure structural dissimilarity. Euclidean distance is defined in terms of the entries of matrix W as:

$$d_{ij} = \sqrt{a_1 \sum_{k \neq i,j} ((w_{ik} - w_{jk})^2 + (w_{ki} - w_{kj})^2) + a_2 (w_{ij} - w_{ji})^2} \tag{7.5}$$

where $a_1 + a_2 = 1$

Correlation coefficient: Correlation coefficient is a measure of structural similarity which in contrast to the Euclidean distance does not directly compare entries in W but their deviations from mean values of rows and columns. Let

$$\bar{w}_{i\bullet} = \frac{1}{n}\sum_{k=1}^{n} w_{ik} \quad \text{resp.} \quad \bar{w}_{\bullet j} = \frac{1}{n}\sum_{k=1}^{n} w_{kj}$$

be the mean of the values of the ith row and jth column of W. The correlation coefficient is defined by

$$c_{ij} = \frac{\sum_{k\neq i,j}(w_{ik} - \bar{w}_{i\bullet})(w_{jk} - \bar{w}_{j\bullet}) + \sum_{k\neq i,j}(w_{ki} - \bar{w}_{\bullet i})(w_{kj} - \bar{w}_{\bullet j})}{\sqrt{\sum_{k\neq i,j}[(w_{ik} - \bar{w}_{i\bullet})^2 + (w_{ki} - w_{\bullet i})^2]}\sqrt{\sum_{k\neq i,j}[(w_{jk} - \bar{w}_{j\bullet})^2 + (w_{kj} - w_{\bullet j})^2]}} \tag{7.6}$$

Note that $c_{ij} \in [-1, 1]$. To get the distance between vertices i and j we combine the absolute value of correlation coefficient with a negation function g. E.g., $d_{ij} = g(|c_{ij}|)$.

Jaccard, Overlap and Cosine indices: When a network is sparse the correlation coefficient is affected by the number of zero elements of matrix W. In this case, more appropriate measures of structural similarity are based on the assumption that two vertices are similar if they share many common neighbors. Let $G = (V, E)$ be a graph with adjacency matrix W and $N(v)$ be the neighbors of a vertex $v \in V$. The common neighbors of vertices v_i and v_j are $N(v_i) \cap N(v_j)$. Then depending on the normalization we have the following similarity indices:

$$\sigma_{Jaccard}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|} \tag{7.7}$$

$$\sigma_{Overlap}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{\min(|N(v_i)|, |N(v_j)|)} \tag{7.8}$$

$$\sigma_{Cosine}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{\sqrt{\sum_{v_k \in V\setminus\{v_i\}} w_{ik}^2} \cdot \sqrt{\sum_{v_k \in V\setminus\{v_j\}} w_{jk}^2}} \tag{7.9}$$

We can write $|N(v_i) \cap N(v_j)|$ in terms of the entries of the adjacency matrix W as $\sum_{v_k \in V\setminus\{v_i,v_j\}} w_{ik}w_{jk}$. Similarly $|N(v_i) \cup N(v_j)| = \sum_{v_k \in V\setminus\{v_i,v_j\}} \max(w_{ik}, w_{jk})$ and $|N(v_i)| = \sum_{v_k \in V\setminus\{v_i\}} w_{ik}$ and $|N(v_j)| = \sum_{v_k \in V\setminus\{v_j\}} w_{jk}$.

When graph G is directed then we consider both the out-neighbors $N^+(v)$ and the in-neighbors $N^-(v)$ of a vertex v. Then we have $|N^+(v_i) \cap N^+(v_j)| + |N^-(v_i) \cap N^-(v_j)| = \sum_{v_k \in V\setminus\{v_i,v_j\}}(w_{ik}w_{jk} + w_{ki}w_{kj})$ and $|N^+(v_i) \cup N^+(v_j)| + |N^-(v_i) \cup N^-(v_j)| = \sum_{v_k \in V\setminus\{v_i,v_j\}}(\max(w_{ik}, w_{jk}) + \max(w_{ki}, w_{kj}))$ and $|N^+(v_i)| = \sum_{v_k \in V\setminus\{v_i\}} w_{ik}$ and $|N^-(v_i)| = \sum_{v_k \in V\setminus\{v_i\}} w_{ki}$ and similarly for $|N^+(v_j)|$ and $|N^-(v_j)|$. Therefore the Jaccard, Overlap and Cosine indices can be written as:

$$\sigma_{Jaccard}(v_i, v_j) = \frac{|N^+(v_i) \cap N^+(v_j)| + |N^-(v_i) \cap N^-(v_j)|}{|N^+(v_i) \cup N^+(v_j)| + |N^-(v_i) \cup N^-(v_j)|} \tag{7.10}$$

$$\sigma_{Overlap}(v_i, v_j) = \frac{|N^+(v_i) \cap N^+(v_j)| + |N^-(v_i) \cap N^-(v_j)|}{\min(|N^+(v_i)|, |N^+(v_j)|) + \min(|N^-(v_i)|, |N^-(v_j)|)} \quad (7.11)$$

$$\sigma_{Cosine}(v_i, v_j) = \frac{|N^+(v_i) \cap N^+(v_j)| + |N^-(v_i) \cap N^-(v_j)|}{\sqrt{\sum_{v_k \in V \setminus \{v_i\}} (w_{ik}^2 + w_{ki}^2)} \cdot \sqrt{\sum_{v_k \in V \setminus \{v_j\}} (w_{jk}^2 + w_{kj}^2)}} \quad (7.12)$$

Also the Jaccard, Overlap and Cosine indices can be extended to take into account the direct links between vertices v_i and v_j. For example:

$$\sigma_{Overlap}(v_i, v_j) = \frac{a_1 \sum_{k \neq i,j} (w_{ik} w_{jk} + w_{ki} w_{kj}) + a_2 (w_{ij} + w_{ji})}{a_1 (\min(s_i^+, s_j^+) + \min(s_i^-, s_j^-)) + a_2 (w_{ij} + w_{ji})} \quad (7.13)$$

where s_i^+ and s_i^- are the out-strength and in-strength of vertex v_i, $a_1 + a_2 = 1$ determine the relative importance of the direct links versus the common neighbors and $w_{ij} = 1$ (resp. $w_{ji} = 1$) if there is an edge between v_i and v_j and 0 otherwise. The generalization of the above measures to greyscale networks is straight forward. Ravasz et al. [162] used the topological overlap to study the hierarchical modular organization of metabolic networks. However the definition of topological overlap has many drawbacks. For example when vertex v_i has edges with value 1 to all other vertices then $\sigma_{Overlap}(v_i, v_j)$ is a constant and is independent of the edges and the edge values of vertex v_j. Note that the same drawback exists in the original definition of topological overlap 7.13 as well as in the generalization proposed by Yip and Horvath [218]. An alternative definition that we propose is:

$$\sigma_{Overlap}(v_i, v_j) = \frac{a_1 \sum_{k \neq i,j} (\min(w_{ik}, w_{jk}) + \min(w_{ki}, w_{kj})) + a_2 (w_{ij} + w_{ji})}{a_1 \sum_{k \neq i,j} (\max(w_{ik}, w_{jk}) + \max(w_{ki}, w_{kj})) + a_2 (w_{ij} + w_{ji})} \quad (7.14)$$

To reduce the noise introduced by taking into account many weak and possibly irrelevant connections in the calculation of topological overlap we can consider only the $p\%$ strongest connections or the connections above a threshold. By combining topological overlap with a negation function g we define the distance between vertices v_i and v_j as: $d_{ij} = g(t_{ij})$

7.4 Vertex Importance Score

Greyscale networks constructed from experimental data may contain as many vertices as is the number of recording channels of the activity of a system. However not all vertices are related to a specific function of the system for which we are interested. One way to distinguish vertices is to assign to them an importance score according a) to the degree to which they have direct and indirect relationships with the other vertices in a network and b) to attributes of the vertices. Although the connection between the importance of a vertex as measured by some metric and its relevance to a specific function is an open research question, there is increasing evidence that important vertices are also relevant [186, 204]. Huang et al. [99] defined a function, which is called *vertex importance score* and measures the importance of a vertex with respect to its relationships and/or its attributes. Vertex importance score is defined as follows:

Definition 7.2 (Vertex Importance Score) *Let $G = (V, E, \omega)$ be a greyscale network, and $R : V \to [0, 1]$ be a function that assigns a real value ranging from 0 to 1 to every vertex v in G. The vertex importance score, $R(v)$, of vertex $v \in V$ is used to rank the importance of vertex v due to its attributes and its role (position) on network G.*

To calculate the vertex importance score $R(v)$ of a vertex v we decomposed it into two terms, $R_{struct}(v)$ and $R_{attrib}(v)$. Term $R_{struct}(v)$ measures the importance of the role of v in a network and is equal to a linear combination of the values of some centrality measures. For example:

$$R_{struct}(v) = a_1 c_S(v) + a_2 c_B(v) + a_3 c_{HBL}(v) \tag{7.15}$$

where $a_1 + a_2 + a_3 = 1$, and the centrality measures $c_S(v)$, $c_B(v)$ and $c_{HBL}(v)$ are defined in Sect. 7.5
Term $R_{attr}(v)$ measures the prestige of vertex v due to attributes of v. For motor imagery EEG data, depending on context, we propose the function:

$$R_{attr}(v) = \frac{\max_{u \in V}(pow_b(u)) - pow_b(v) + \varepsilon}{\max_{u \in V}(pow_b(u)) - \min_{u \in V}(pow_b(u)) + \varepsilon} \tag{7.16}$$

or the function

$$R_{attr}(v) = \frac{pow_b(v) - \min_{u \in V}(pow_b(u)) + \varepsilon}{\max_{u \in V}(pow_b(u)) - \min_{u \in V}(pow_b(u)) + \varepsilon} \tag{7.17}$$

where $pow_b(v)$ is the power spectrum (average) at band $b \in \{mu, beta\}$ of the signal that corresponds to vertex v and ε is a small number. To sum up the vertex importance score is defined as:

$$R(v) = R_{struct}(v) \cdot R_{attr}(v) \tag{7.18}$$

The decomposition of $R(v)$ into two terms is a heuristic rule that seems to work when applied to data from brain activity measurements. However one can easily devise other rules appropriate for his/her field of study. Note that the Hubbell's centrality measure c_{HBL} [100] already combines information concerning the role of a vertex in a network with information concerning the prestige of the vertex.

Having ranked the vertices, we may either highlight the important ones or filter out the less important ones. A filtered network can be produced by removing all the vertices that their vertex importance score R is under a threshold θ which is specified by users. The set of the remaining vertices after filtering is

$$V' = \{v \mid v \in V \ \wedge \ R(v) \geq \theta\}$$

7.5 Centrality Measures

Characterizing local properties of networks is important both in theory and practice since at the local scale we can detect which vertices are the most relevant for the organization and functioning of a network. These local measures are commonly named centrality measures (or centrality indices) and have proved of great value in analyzing the role played by individuals in social networks and in identifying essential proteins [219], keystone species [103], and functionally important brain regions [87].

There is no commonly accepted definition for centrality. Many authors introduced their own centrality, avoiding a strict definition for centrality in general. The intuition about a centrality measure is that it denotes an order of importance on the vertices or edges of a graph by assigning real values to them. As a minimal requirement, most centrality measures depend only on the structure of a graph. Another issue is that the formulas for these different measures make implicit assumptions about the manner in which things flow in a network [41]. For example, some measures, such as Freeman's closeness and betweenness [73], count only geodesic paths, apparently assuming that whatever flows through the network moves only along the shortest possible paths. Other measures, such as flow betweenness (Freeman et al. [74]), do not assume shortest paths, but do assume proper paths in which no vertex is visited more than once. Still other measures, such as Bonacich's [39] eigenvector centrality and Hubbell's centrality [100], count walks, which assume that trajectories can not only be circuitous, but also revisit vertices and edges multiple times along the way. Regardless of trajectory, some measures (e.g., betweenness) assume that what flows from vertex to vertex is indivisible (like a package) and must take one path or another, whereas other measures (e.g., eigenvector) assume multiple "paths" simultaneously (like information or infections). In subsequent sections will be presented some centrality measures that seems to be promising in analyzing brain networks. Their extension to greyscale networks will also be discussed.

7.5.1 Centrality Measures Based on Neighborhoods

Degree Centrality

The simplest centrality is the degree centrality $c_D(v)$ of a vertex v that is simply defined as the degree $deg(v)$ of v if the considered graph is undirected. In directed networks two variants of the degree centrality may be appropriate: the in-degree centrality $c_{iD}(v) = deg^-(v)$ and the out-degree centrality $c_{oD}(v) = deg^+(v)$. Degree centrality can be normalized to range from 0 to 1, where 0 means the smallest possible and 1 the highest possible centrality. The

normalized measures are called relative measures of centrality:

$$c_D(v) = \frac{deg(v)}{n-1} \tag{7.19}$$

$$c_{iD}(v) = \frac{deg^-(v)}{n-1} \tag{7.20}$$

$$c_{oD}(v) = \frac{deg^+(v)}{n-1} \tag{7.21}$$

The degree centrality is a local measure because the centrality value of a vertex is only determined by the number of its neighbors. However the degree is the most fundamental network measure and most other centrality measures are linked to it. The degree sequence is argued to reflect some fundamental aspects of biological, social and technological networks [4].

Strength Centrality

Degree centrality can easily be extended to greyscale networks. In symmetric networks, the strength centrality $c_S(v)$ is equal to the strength $s(v)$ of vertex v. In asymmetric networks the in-strength centrality $c_{iS}(v)$ is equal to in-strength $s^-(v)$ and the out-strength centrality $c_{oS}(v)$ is equal to out-strength $s^+(v)$ of vertex v. The corresponding normalized measures are:

$$c_S(v) = \frac{s(v)}{n-1} = \frac{1}{n-1} \sum_{e=\{v,u\}\in E} \omega(e) \tag{7.22}$$

$$c_{iS}(v) = \frac{s^-(v)}{n-1} = \frac{1}{n-1} \sum_{e=(u,v)\in E} \omega(e) \tag{7.23}$$

$$c_{oS}(v) = \frac{s^+(v)}{n-1} = \frac{1}{n-1} \sum_{e=(v,u)\in E} \omega(e) \tag{7.24}$$

7.5.2 Centrality Measures Based on Distances

Shortest-Path Closeness Centrality

Closeness can be regarded as a measure of how long it will take information to spread from a given vertex to others in the network. Let $G = (V, E)$ be an undirected graph. Then the shortest path closeness centrality of vertex $v \in V$ is defined as the inverse of the mean geodesic distance from vertex v to every other vertex reachable from it [28, 73].

$$c_C(v) = \frac{n_C}{\sum_{u\neq v} d_G(v,u)} \tag{7.25}$$

where $d_G(v,u)$ denotes the length of a shortest path between vertices v and u and $n_C = n-1$ is a normalizing constant. Thus the above equation measures the degree to which a vertex is close to other vertices (on average). (7.25) can be used in directed graphs without modifications. However one serious drawback of this equation is that it can only be used for connected graphs (resp. strongly connected digraphs).

The generalization of closeness centrality to greyscale networks is straightforward (see Sect. 7.3.1).

Shortest-Path Efficiency

In Latora and Marchiori [118] a new measure, called efficiency, is given.

$$c_{Ef}(v) = \frac{1}{n_{Ef}} \sum_{u \neq v} \frac{1}{d_G(v,u)} \tag{7.26}$$

where $n_{Ef} = n - 1$.
This measure can also be interpreted as closeness centrality since the quantity $\frac{1}{d_G(v,u)}$ can be considered as closeness between vertices v and u and $C_{Ef}(v)$ is the mean of this quantity for $u \in V \setminus \{v\}$. Note that (7.26) can also be used for disconnected graphs. If some vertices v and u are not connected then they do not contribute to $C_{Ef}(v)$ ($d_G(v,u) = +\infty \Rightarrow \frac{1}{d_G(v,u)} = 0$).

The generalization of closeness centrality to greyscale networks is straightforward (see Sect. 7.3.1).

Shortest-Path Betweenness Centrality

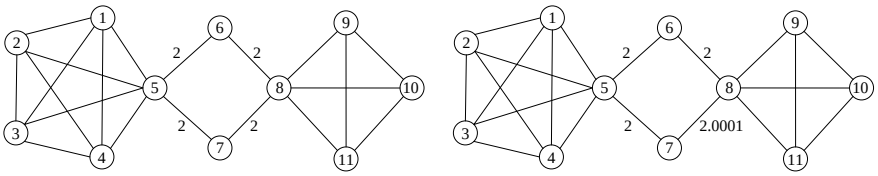
Denote by σ_{st} the number of shortest (s,t)-paths, and let $\sigma_{st}(v)$ be the number of shortest (s,t)-paths passing through some vertex v other than s, t. Then, the shortest-path betweenness $c_B(v)$ of a vertex $v \in V$ is defined to be

$$c_B(v) = \frac{1}{n_B} \sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v,s\}} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{7.27}$$

where $n_B = (n-1)(n-2)$ is a normalizing constant. The shortest path betweenness centrality sums up the relative number of shortest paths that pass through vertex v for each pair $s, t \in V \setminus \{v\}$, $s \neq t$ of end vertices. The relative numbers $\frac{\sigma_{st}(v)}{\sigma_{st}}$ are interpreted as the extend to which vertex v controls the communication between vertices s and t. A vertex is central, if it is between many pairs of other vertices. The definition of betweenness applies to disconnected graphs without modification. The number of shortest paths between two vertices not connected by a directed path is zero, and the resulting zero contribution is exactly what is desired.

Brandes [44] showed how shortest-path betweenness centrality can be generalized to greyscale networks. He proposed two different generalizations.

The first generalization of shortest paths to greyscale networks consists of the definition of the length of a path as the sum of the length of its edges 7.1. This generalization is referred to as weighted betweenness in Brandes (2001) [43]. With edge lengths, the algorithm proposed by Brandes [44] can be applied, but may lead to counter-intuitive results a) because the calculated lengths might not reflect the intuition of distance between two vertices and b) due to the following problem. When there are many alternative shortest paths that connect two vertices u and v, then, in general, only one of them will be used for transportation between u and v. Therefore the vertices of this shortest path control the communication between u and v and will receive a full credit in the calculation of betweenness centrality while the vertices in the alternative shortest paths will receive no credit at all. This problem is shown in Fig. 7.5. In Fig. 7.5(a) and Fig. 7.5(b) all drawn edges are assigned positive values which are interpreted as distances. In Fig. 7.5(a), edges $(5,6)$, $(5,7)$, $(6,8)$ and $(7,8)$ have equal values



(a) Network A (b) Network B

Figure 7.5: Shortest-path betweenness centrality in valued networks. Vertices 1, 2, 3, 4, 9, 10, and 11 have zero shortest-path betweenness centrality in both networks. In network A, vertices 5, 8, 6, and 7 have centralities 0.544, 0.478, 0.222, and 0.222 respectively. In network B, vertices 5, 8, 6, and 7 have centralities 0.533, 0.467, 0.422, and 0 respectively.

(e.g. 2), but in Fig. 7.5(b) the value of edge $(7, 8)$ differs by a tiny amount from the values of edges $(5, 6)$, $(5, 7)$, $(6, 8)$ (e.g. 0.0001). Then, we see that in Fig. 7.5(a) the shortest paths from vertices 1 ,2, 3, 4, and 5 to vertices 8, 9, 10, 11 pass either from vertex 6 or from vertex 7. However in Fig. 7.5(b) all these shortest paths pass from vertex 6 none of them pass from vertex 7.

An alternative generalization of shortest-path betweenness centrality uses the edge values in the definition of the relative weight rather than the length of a path 7.2. We applied this generalization and the corresponding algorithm proposed by Brandes [44], to greyscale networks and we identified two potential problems; a) numerical overflows/underflows and b) unsuitability of the approach for dense networks with variable edge weights.

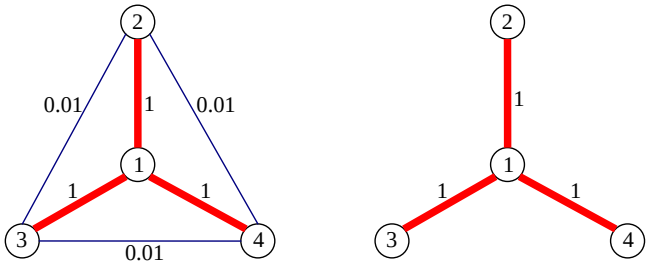
Numerical overflows/underflows may occur due to repeated multiplications of edge weights through long paths. This problem can be alleviated with scaling of edge weights and/or numerical techniques which are beyond the scope of this study.

The second generalization of betweenness uses the structure of the network to find the shortest paths between two vertices and then uses the edge values to weight these paths. This approach fails to produce meaningful results when applied to networks that are almost complete but have big variations in edge weights. For example, in Fig. 7.6(a) the centrality of vertex 1 is zero although intuitively it is the center of the network. The results are improved if we clean the edges with low values (Fig. 7.6(b)) by applying an adjacency function 6.6. Then the centrality of vertex 1 is one as we expected.

7.5.3 Spectral Centrality Measures

Bonacich's Eigenvector Centrality

In 1972 Phillip Bonacich introduced a centrality measure based on the eigenvectors of adjacency matrices [38]. He presented three different approaches for the calculation and all three of them result in the same valuation of the vertices. The three methods of calculation are: a) the factor analysis, b) the convergence of an infinite sequence, and c) the solving of a linear equation system. We will describe the third method because of its simplicity and popularity.



(a) A clique with 4 vertices

(b) After removing week links the clique becomes a star

Figure 7.6: The second generalization of shortest-path betweenness centrality in valued networks. Vertices 2, 3, and 4 have zero shortest-path betweenness centrality as expected. However vertex 1 has also zero betweenness contrary to the fact that is the center of the network. If we apply a cutoff function to remove the edges with week values then the normalized centrality of vertex 1 is one.

The centrality of each vertex is proportional to the sum of the centralities of the vertices to which it is directly connected.

$$\lambda c(v_i) = \sum_{j=1}^{n} w_{ji} c(v_j)$$

In matrix notation with $\mathbf{c} = [c(v_1), c(v_2), \ldots, c(v_n)]^T$ this yields:

$$\lambda \mathbf{c} = W^T \mathbf{c} \tag{7.28}$$

This type of equation is well known and solved by the eigenvalues and eigenvectors of W^T. From the set of different eigenvectors only one seems to be an appropriate solution that can serve as a centrality measure (see, e.g. Bonacich [38]). Since the entries of matrix W are nonnegative $w_{ij} \geq 0$ the Perron-Frobenius theorem 7.3 holds.

We call the eigenvector $\mathbf{s} = [s_1, \ldots, s_n]^T$ of the maximal eigenvalue of (7.28) principal eigenvector. Then the eigenvector centrality is of node v_i is defined as

$$c_{EV}(v_i) = \frac{|s_i|}{\| \mathbf{s} \|_p} \tag{7.29}$$

where the centrality vector $\mathbf{s}$ is normalized by dividing it by its p-norm.

$$\| \mathbf{s} \|_p = \begin{cases} (\sum_{i=1}^{n} |s_i|^p)^{\frac{1}{p}} & 1 \leq p < \infty \\ max_{i=1,\ldots,n}\{|s_i|\} & p = \infty \end{cases}$$

to produce centrality scores $c(v_i) \leq 1$.

Katz's status

As a generalization of simply using in-degrees to measure "status" in social networks, Katz [109] suggested that the prominence of a vertex is determined

by the number of directed walks of arbitrary length ending in the vertex, where the influence of longer paths is attenuated by a decay factor. In particular, Katz suggests that we measure influence by the column sums of the matrix $\sum_{k=1}^{\infty} \alpha^k A^k$. Therefore the status of vertex v_i is

$$\mathbf{c}(v_i) = \sum_{j=1}^{n} (\sum_{k=1}^{\infty} \alpha^k A^k)_{ji}$$

where α is an "attenuation" factor.
The infinite sum $\sum_{k=1}^{\infty} \alpha^k A^k$ converges to $(I - \alpha A)^{-1} - I$ as long as $|\alpha| < 1/\lambda_1$, where λ_1 is the maximum value of an eigenvalue of A. Finally, in matrix notation we have

$$\mathbf{c} = \sum_{k=1}^{\infty} a^k (A^T)^k \cdot \mathbf{1}_n = ((I - \alpha A^T)^{-1} - I) \cdot \mathbf{1}_n \tag{7.30}$$

Therefore

$$\mathbf{c} = (I - \alpha A^T)^{-1} \cdot \alpha \cdot \mathbf{deg}^- \tag{7.31}$$

where $\mathbf{deg}^-$ is the vector of in-degrees.

The generalization of the status index of Katz to greyscale networks is done by substituting the value matrix W for the adjacency matrix A. Note that the entries of the k-th power of the adjacency matrix of a graph give the number of paths of length k between every pair of vertices, whereas the k-th power of the value matrix W give the total weight of all paths of length k.

From Katz's index we get a normalized centrality measure by dividing $\mathbf{c}$ by its norm

$$\mathbf{c}_K = \frac{\mathbf{c}}{\| \mathbf{c} \|_p} \tag{7.32}$$

Hubbell's Centrality

Charles Hubbell [100] suggested a centrality measure based on the solution of a system of linear equations. His approach uses directed valued graphs where the weights of the edges may be real numbers. The general assumption of Hubbell's centrality is similar to the idea of Bonacich, but the centrality of a vertex depends both on its connection to other vertices and to exogenous input which sometimes is called boundary conditions. For example the importance of an ECoG channel for a specific psychological experiment, depends both on how much synchronized this channel is with other channels and on the power spectral density of the signal at specific frequency bands.

Letting e be a vector of these exogenous sources of status or information, we could replace (7.28) with a new equation.

$$\mathbf{c} = \alpha W^T \mathbf{c} + \mathbf{e} \tag{7.33}$$

where $\mathbf{c} = [c(v_1), c(v_2), \ldots, c(v_n)]^T$ and $\mathbf{e} = [e_1, e_2 \ldots, e_n]^T$. The parameter α reflects the relative importance of endogenous versus exogenous factors in the determination of centrality. As in Katz's index in order to get meaningful results α should be chosen according to restriction $|\alpha| < 1/\lambda_1$, where λ_1 is the maximum value of an eigenvalue of W (see Fig. 7.7 for an example). This restriction is not mentioned in the literature.
(7.35) has the solution:

$$\mathbf{c} = (I - \alpha W^T)^{-1} \mathbf{e} \tag{7.34}$$

Then the Hubbell's centrality is defined as:

$$\mathbf{c}_{HBL} = \frac{\mathbf{c}}{\| \mathbf{c} \|_p} \tag{7.35}$$

Page Rank

PageRank was presented by Page et al. in 1999 [155]. The main idea is to score a Web page with respect to its topological properties. PageRank is a feedback centrality since the centrality of a web page depends on the number and centrality of Web pages linking to it. In matrix notation PageRank can be written as:

$$\mathbf{c} = \alpha(S^+)^{-1}W^T\mathbf{c} + (1-\alpha)\mathbf{1}_n \tag{7.36}$$

where $0 < \alpha < 1$, $\mathbf{1}_n$ is the n-dimensional vector that has all its entries equal to 1 and S^+ is the diagonal matrix where the i-th entry contains the out-strength of vertex v_i (i.e., $S^+_{ii} = c_{oS}(v_i) = \sum_{v_k \in V} w_{ik}$, for $i = 1, \ldots, n$)

Shortly after the presentation of PageRank numerous modifications and application of it appeared in the literature. One interesting modification and application of PageRank is in the study of food webs [10]. These are the complex networks of who eats whom in an ecosystem. In PageRank, a web page is important if important pages point to it. In food webs a species is important if it points to important species. GeneRank is another intuitive modification of PageRank that maintains many of its mathematical properties. It combines gene expression information with a network structure derived from gene annotations (gene ontologies) or expression profile correlations [139].

Comparison of Eigenvector, Katz's, Hubbell's and PageRank Centralities

By comparing (7.28), (7.31), (7.34) and (7.36) we see that the eigenvector, Katz's, Hubbell's and PageRank centralities have a similar form, namely:

$\mathbf{c} = \alpha \cdot W^T\mathbf{c}$	where	$\alpha = \frac{1}{\lambda_1}$	(eigenvector)
$\mathbf{c} = \alpha \cdot W^T\mathbf{c} + \alpha \cdot \mathbf{deg}^-$	where	$\lvert\alpha\rvert < \frac{1}{\lambda_1}$	(Katz's)
$\mathbf{c} = \alpha \cdot W^T\mathbf{c} + \mathbf{e}$	where	$\lvert\alpha\rvert < \frac{1}{\lambda_1}$	(Hubbell's)
$\mathbf{c} = \alpha(S^+)^{-1}W^T\mathbf{c} + (1-\alpha)\mathbf{1}_n$	where	$0 < \alpha < 1$	(PageRank)

Subgraph Centrality

Subgraph centrality has been introduced by Estrada et al. [69] and is defined as a weighted sum of the number of closed walks in a graph, where longer walks receive lower weight than shorter ones. The number of closed walks, $\mu_k(i)$, of length k starting and ending on vertex v_i is given by the ith diagonal entry of the kth power of the adjacency matrix, A

$$\mu_k(i) = (A^k)_{ii}$$

These closed walks are directly related to the subgraphs of the network. Then the subgraph centrality of vertex v_i is defined as:

$$c_{SG}(v_i) = \sum_{k=0}^{\infty} \frac{\mu_k(i)}{k!} \tag{7.37}$$

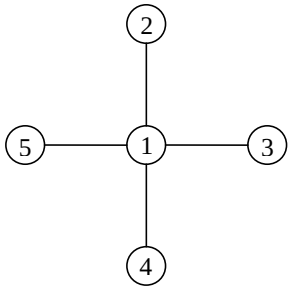


Figure 7.7: Star network with 5 vertices. The largest eigenvalue of the adjacency matrix is $\lambda_1 = 2$. If we choose $\alpha = 0.55 > 1/\lambda_1$ then $\mathbf{c} = [-15.2381, -7.3810, -7.3810, -7.3810, -7.3810]^T$. If we choose $\alpha = 0.45 < 1/\lambda_1$ then $\mathbf{c} = [14.7368, 7.6316, 7.6316, 7.6316, 7.6316]^T$ which makes more sense.

Estrada et al. [69] proved that for undirected networks (7.37) becomes:

$$c_{SG}(v_i) = \sum_{j=1}^{n} q_j(i)^2 e^{\lambda j} \tag{7.38}$$

where $\lambda_1, \ldots, \lambda_n$ and $q_1, \ldots, q_n$ are the eigenvalues and the corresponding eigenvectors of A, and $q_j(i)$ denotes the ith entry of q_j. In matrix notation (7.38) can be written as:

$$\mathbf{c}_{SG} = diag(Qe^{\Lambda}Q^T) \tag{7.39}$$

where $A = Q\Lambda Q^T$, $\Lambda = diag([\lambda_1, \ldots, \lambda_n])$, $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ and

$$e^{\Lambda} = \begin{pmatrix} e^{\lambda_1} & 0 & \ldots & 0 \\ 0 & e^{\lambda_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & e^{\lambda_n} \end{pmatrix}$$

This measure generalizes to greyscale networks by substituting matrix W for A.

Functional Centrality

Functional centrality has been introduced by Rodríguez et al. [164] as a generalization of subgraph centrality. The framework behind this measure seems to unify a number of centrality measures including the degree centrality and the Katz's status index. Let λ be the spectral radius of a graph G. Let f be a function, $f : (R) \to \mathbb{R}$, whose Taylor's series is $f(x) = \sum_{k=0}^{\infty} \alpha_k x^k$, $|x| < r$, where $r > \lambda$. The functional centrality, $c_f(v_i)$, is defined as:

$$c_f(v_i) = \sum_{k=0}^{\infty} \alpha_k \mu_k(i) \tag{7.40}$$

where $\mu_k(i)$ denotes the closed walks of length k that start and end on vertex v_i. When the network is undirected then matrix W has n nonnegative eigenvalues. In matrix notation (7.41) can be written as:

$$\mathbf{c}_f = diag(Qf(\Lambda)Q^T) \tag{7.41}$$

where $W = Q\Lambda Q^T$, $\Lambda = diag([\lambda_1, \ldots, \lambda_n])$, $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ and

$$f(\Lambda) = \begin{pmatrix} f(\lambda_1) & 0 & \ldots & 0 \\ 0 & f(\lambda_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & f(\lambda_n) \end{pmatrix}$$

7.6 Network Level Measures

Networks are often classified into unifying categories in order to obtain a better understanding of their structure and function. In order to classify a network it is necessary to reduce the information to describe essential properties of the network into a single number or a few numbers. This number or numbers are called *network measures* or *network statistics*. Network measures should catch the relevant and needed information, they should differentiate between certain classes of networks and should be easily computed in order to be useful in algorithms and applications.

7.6.1 Average Degree

The average degree, K, is an easily computed statistic.

$$K = \frac{1}{n} \sum_{v \in V} deg(v) \tag{7.42}$$

7.6.2 Network Centrality

Vertex centralities provide insight into the vertices' location in the network. The relationship between the centralities of all vertices can reveal aspects of the overall network structure. Intuitively, some networks are more centralized than others, i.e. some networks are more depending on the most central vertices than others. A very centralized network is dominated by one or a few very central vertices. If these vertices are removed or damaged, the network quickly fragments into unconnected sub-networks. The star topology in which only one vertex v is connected to all others but all vertices are only connected to v is a very centralized network. A clique where every vertex is connected to every other vertex is not centralized.

If we want to measure the degree to which the graph as a whole is centralized, we look at the dispersion of a vertex centrality measure. Freeman [72] has proposed a very general approach with which the centralization $c_X(G)$ of a network G can be calculated in relation to the values of any normalized vertex centrality index c_X:

$$c_X(G) = \frac{1}{n-1} \sum_{v \in V} (c_X(u)^* - c_X(v)) \tag{7.43}$$

where $c_X(u)^*$ denotes the largest centrality value that occurs across the vertices in the network under investigation; that is $c_X(u)^* = \max_{v \in V} c_X(v)$. This approach measures the average difference in centrality between the most central vertex and all others. The network centralization value is in the range $[0, 1]$, since normalized vertex centralities in the range of $[0, 1]$ are used. Other obvious possibilities to generate a network index from the distribution of vertex centrality indices are to compute the variance of the values (see (7.44))or the maximal difference between centrality values or any other statistics on these values.

$$\sigma_X^2(G) = \frac{1}{n} \sum_{u \in V} (c_X(u) - \bar{c}_X)^2 \tag{7.44}$$

where $\bar{c}_X = \sum_{u \in V} c_X(u)$

7.6.3 Clustering Coefficient and Transitivity

The *clustering coefficient* introduced by Watts and Strogatz [211] in 1998. For a vertex v the clustering coefficient $c(v)$ measures the connectivity of its direct neighborhood. The clustering coefficient $C(G)$ of a graph is the average of $c(v)$ taken over all vertices. In 2000 Barrat and Weigt [25] used an alternative formulation of this average. In 2002 Newman, Strogatz and Watts [147] introduced the *transitivity* as an alternative formulation of the clustering coefficient of a graph. Transitivity turned out to be equivalent to the formulation of Barrat and Weigt and not to the original clustering coefficient.

We will define both measures in terms of triangles and triples. Let $G = (V, E)$ be an undirected simple graph, then a *triangle* is a complete subgraph of G with exactly three vertices. The *number of triangles of vertex* v is defined as

$$\delta(v) = |\{\{u, w\} \in E \ : \ \{v, u\} \in E \text{ and } \{v, w\} \in E\}|$$

Since each triangle contains three vertices (hence if we sum over all vertices each triangle is counted three times) the number of triangles of graph G is

$$\delta(G) = \frac{1}{3} \sum_{v \in V} \delta(v) \tag{7.45}$$

A *triple* at a vertex v is a path of length two for which v is the center vertex. The *number of triples of vertex* v is then defined as

$$\tau(v) = \binom{deg(v)}{2} \quad ,$$

and summing the triples of all vertices we have the triples for the whole graph

$$\tau(G) = \sum_{v \in V} \tau(v) \tag{7.46}$$

With the introduced notation the clustering coefficient for a vertex v with degree $deg(v) \geq 2$ is defined as

$$c(v) = \frac{\delta(v)}{\tau(v)} \tag{7.47}$$

The clustering coefficient $C(G)$ of the whole graph is the average over the clustering coefficients of its vertices

$$C(G) = \frac{1}{|V'|} \sum_{v \in V'} c(v) \tag{7.48}$$

where $V' = \{v \in V \mid deg(v) \geq 2\}$. Alternatively to (7.48) the term $c(v)$ is sometimes defined to be either zero or one for vertices v with degree at most 1 and included in the sum. Note that the choice of the definition is important.

The transitivity of a graph G is defined as the average over the vertex clustering coefficients weighted by the denominators, $\tau(v)$, of (7.47)

$$T(G) = \frac{1}{\sum_{v \in V'} \tau(v)} \sum_{v \in V'} \tau(v) \cdot c(v) \quad \text{(Bollobás et al. [37])} \tag{7.49}$$

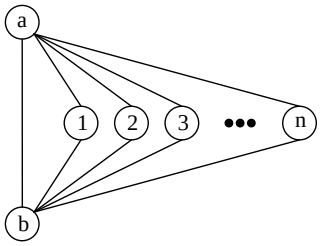


Figure 7.8: The above network has $n+2$ vertices in total: Vertices $1, 2, \ldots, n$, plus other two vertices a and b. The transitivity for this network is $T(G) = \frac{3n}{\binom{n+1}{2}+\binom{n+1}{2}+1+\ldots+1} = \frac{3}{n+2}$, which converges to 0 as $n \to \infty$. On the other hand, the clustering coefficient is $C(G) = \frac{1}{n+2}(\frac{2}{n+1} + \frac{2}{n+1} + 1 + \ldots + 1) = \frac{n^2+n+4}{n^2+3n+2}$, which converges to 1 when $n \to \infty$

$$\Rightarrow T(G) = \frac{3\delta(G)}{\tau(G)} \quad \text{(Newman et al. [147])} \tag{7.50}$$

As there are exactly three triples in each triangle $3\delta(G) \leq \tau(G)$ holds and consequently $T(G)$ is a rational number between zero and one.

Comparing the two measures, clustering coefficient C calculates the mean of the ratio, whereas transitivity T the ratio of the means. Clustering coefficient C tends to weight the contribution of low degree vertices more heavily, because such vertices have a small denominator. While in some cases C is indeed close to T, in other cases the two measures give totally different results [119] (see Fig. 7.8).

Recently, a number of authors have attempted to generalize the clustering coefficient concept to the case of valued networks (Barrat et al. [24]; Lopez-Fernandez et al. [125]; Onnela et al. [150]; Zhang and Horvath [220]), producing a range of possible definitions.

Onnela et al. [150] proposed a version of local clustering coefficient based on the concept of subgraph intensity, defined as the geometric average of subgraph edge values, resulting in:

$$c_O(v) = \frac{1}{\binom{deg(v)}{2}} \sum_{i \neq j \in V \setminus \{v\}} (\hat{w}_{vi}\hat{w}_{ij}\hat{w}_{jv})^{\frac{1}{3}} \Rightarrow$$

$$c_O(v) = \frac{1}{\max_{i,j}(w_{ij})\binom{deg(v)}{2}} \sum_{i \neq j \in V \setminus \{v\}} (w_{vi}w_{ij}w_{jv})^{\frac{1}{3}} \tag{7.51}$$

Here the edge values are normalized by the maximum value in the network, $\hat{w}_{ij} = w_{ij}/\max_{l,k} w_{lk}$, and the contribution of each triangle depends on all of its edge values. Thus triangles in which each edge value equals $\max_{l,k} w_{lk}$ contribute unity to the sum, while a triangle having one link with a negligible value will have a negligible contribution to the local clustering coefficient. Note that the above clustering coefficient definition as well as the definitions proposed by Barrat et al. [24] and Lopez-Fernandez et al. [125] suffer from the drawback that they require an underlying binary network; if this is not available as a separate set of data, then presumably it must be obtained by discretizing the

valued edges. Hence, as in the case where the original binary definition is used for weighted networks, they are dependent upon some threshold parameter. Further, they break down in the case where the number of binary neighbors of vertex v is less than 2.

A definition that uses only the network values was proposed by Zhang and Horvath [220] in the context of gene coexpression networks.

$$c_Z(v) = \frac{\sum_{i \neq j \in V \setminus \{v\}} \hat{w}_{vi} \hat{w}_{ij} \hat{w}_{jv}}{\sum_{i \neq j \in V \setminus \{v\}} \hat{w}_{vi} \hat{w}_{jv}} \Rightarrow$$

$$c_Z(v) = \frac{1}{\max_{i,j}(w_{ij})} \cdot \frac{\sum_{i \neq j \in V \setminus \{v\}} w_{vi} w_{ij} w_{jv}}{\sum_{i \neq j \in V \setminus \{v\}} w_{vi} w_{jv}} \tag{7.52}$$

where the weights have again been normalized by $\max_{i,j}(w_{ij})$ as above. Note that this measure is identical to the one proposed by Holme et al. [91], and similar to the one (without the normalizing factor $\max_{i,j}(w_{ij})$) proposed by Ahnert et al. [2] when the edge values are interpreted as probabilities. Also this measure is a natural extension of the Watts-Strogatz vertex clustering coefficient since

$$c(v) = \frac{\delta(v)}{\tau(v)} = \frac{\sum_{i \neq j \in V \setminus \{v\}} a_{vi} a_{ij} a_{jv}}{\sum_{i \neq j \in V \setminus \{v\}} a_{vi} a_{jv}}$$

where $A = (a_{ij})$ is the adjacency matrix of a graph.

Having defined the local (vertex) clustering coefficient, the graph clustering coefficient can be defined as the average (7.48) of vertex clustering coefficients. In case that there is a vertex value function $\rho : V \to \mathbb{R}^+$ that maps each vertex to a positive number the graph clustering coefficient may be defined as [175]

$$C(G) = \frac{1}{\sum_{v \in V'} \rho(v)} \cdot \sum_{v \in V'} \rho(v) c(v) \tag{7.53}$$

Equation (7.53) can be used to define the transitivity of a valued network in terms of value matrix W. Inspired by (7.49) and by choosing $\rho(v) = \sum_{i \neq j \in V \setminus \{v\}} w_{vi} w_{jv}$ we have

$$T(G) = \frac{1}{\max_{i,j}(w_{ij})} \cdot \frac{\sum_{v \in V} \sum_{i \neq j \in V \setminus \{v\}} w_{vi} w_{ij} w_{jv}}{\sum_{v \in V} \sum_{i \neq j \in V \setminus \{v\}} w_{vi} w_{jv}} \tag{7.54}$$

Due to the fact that $w_{ii} = 0$ for $i = 1, \ldots, n$, the above equation can be written in matrix notation as

$$T(G) = \frac{1}{\max_{i,j}(w_{ij})} \cdot \frac{trace(W^3)}{trace(W^2(J_n - I_n))}$$

where I_n is the $n \times n$ identity matrix and J_n is the $n \times n$ unit matrix consisting of all ones.

7.6.4 Assortative Mixing

One important network feature, is the similarity between properties of adjacent network vertices. Newman [144] proposed an interesting measure to quantify

the degree of similarity (dissimilarity) between adjacent vertices in a network using *assortative mixing*, which is given by correlation between properties of every pairs of adjacent vertices. Each vertex may have assigned a single scalar or a set of scalar properties. Suppose, for simplicity, that each vertex $v \in V$ is assigned a scalar property $\rho : V \to \mathbb{R}$, such as age or race in social networks or a centrality measure of the vertex position in a network. Then the assortativity coefficient for an undirected graph is defined as the (sample) Pearson product-moment correlation coefficient written in a symmetrical form [144].

$$r = \frac{4m \sum_{\{u,v\}\in E} \rho(u)\rho(v) - \left[\sum_{\{u,v\}\in E}(\rho(u) + \rho(v))\right]^2}{2m \sum_{\{u,v\}\in E}(\rho(u)^2 + \rho(v)^2) - \left[\sum_{\{u,v\}\in E}(\rho(u) + \rho(v))\right]^2} \tag{7.55}$$

The assortativity coefficient for directed graphs can be calculated by simply ignoring the direction of edges and using the above equation. However a more meaningful result for a directed graph is obtained by using the following equation [145]

$$r = \frac{m \sum_{(u,v)\in E} \rho(u)\rho(v) - AB}{\sqrt{m \sum_{(u,v)\in E} \rho(u)^2 - A^2}\sqrt{m \sum_{(u,v)\in E} \rho(v)^2 - B^2}} \tag{7.56}$$

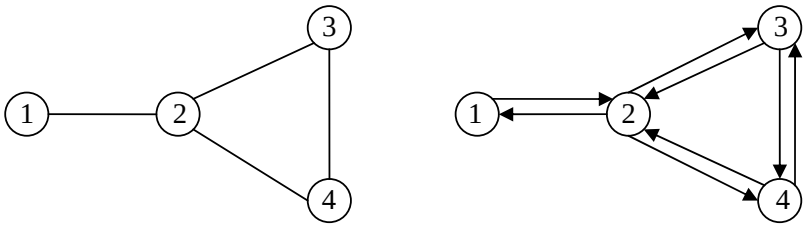
where $A = \sum_{(u,v)\in E} \rho(u)$ and $B = \sum_{(u,v)\in E} \rho(v)$.
The above equation can also be used for undirected graphs if we substitute each undirected edge $\{u, v\}$ with two directed edges (u, v) and (v, u) (see Fig. 7.9 for an example). However, for undirected graphs, (7.57) produces the same result using with fewer arithmetic operations.

The value of r lies in the range $-1 \leq r \leq 1$, with $r = 1$ indicating perfect assortativity and $r = -1$ indicating perfect disassortativity (perfect negative correlation between the properties of the endvertices of edges of the graph). Social, enzyme interaction, protein contact networks and brain functional networks tend to be assortative while technological, protein interaction, metabolic signaling pathways, gene regulatory and brain anatomical networks tend to be disassortative [146, 156, 66]. Disassortative mixing observed in certain biological networks is conjectured to be responsible for decreasing the likelihood of crosstalk between different functional modules of the cell, and increasing the overall robustness of a network by localizing effects of deleterious perturbations [131]. On the other hand, from computational studies, it has been observed [144, 145, 216] that information gets easily transferred through assortative networks as compared to that in disassortative networks.

Leung et al. [121] extended the definition of the assortativity coefficient to cover the class of undirected valued networks.

$$r = \frac{4H \sum\limits_{\{u,v\}\in E} \omega(u,v)\rho(u)\rho(v) - \left[\sum\limits_{\{u,v\}\in E} \omega(u,v)(\rho(u) + \rho(v))\right]^2}{2H \sum\limits_{\{u,v\}\in E} \omega(u,v)(\rho(u)^2 + \rho(v)^2) - \left[\sum\limits_{\{u,v\}\in E} \omega(u,v)(\rho(u) + \rho(v))\right]^2} \tag{7.57}$$

where $H = \sum_{e\in E} \omega(e)$ is the sum of all values of edges in E.



(a) An undirected graph

(b) After substituting each edge with two directed edges

Figure 7.9: An example of how to use (7.56) for undirected graphs. Each undirected edge is substituted with two directed edges pointing in opposite directions. The set of edges is: $E = \{(1,2),\ (2,1),\ (2,3),\ (3,2),\ (2,4),\ (4,2),\ (3,4),\ (4,3)\}$. Suppose that we chose function ρ to be the degree of the directed graph ($\rho \equiv \deg$). Then the list of degrees that correspond to origin vertices of edges in E is: $x = [2,6,6,4,6,4,4,4]$. Similarly the list of degrees that correspond to destination vertices of edges in E is: $y = [6,2,4,6,4,6,4,4]$. Then the assortativity coefficient is $r = \text{corr}(x,y) = -0.714286$

The above equation can also be written in terms of the entries of matrix W as:

$$r_w = \frac{4H\sum_{i<j} w_{ij}\rho(v_i)\rho(v_j) - [\sum_{i<j} w_{ij}(\rho(v_i)+\rho(v_j))]^2}{2H\sum_{i<j} w_{ij}(\rho(v_i)^2+\rho(v_j)^2) - [\sum_{i<j} w_{ij}(\rho(v_i)+\rho(v_j))]^2} \tag{7.58}$$

where the indices i and j take values from 1 to n and from $i+1$ to n respectively, $V = \{v_1,\ldots,v_n\}$ is the set of vertices, and $H = \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}$ is the sum of all values of the upper triangular part of matrix W.

We propose the following generalization of the assortativity coefficient to directed valued networks.

$$r = \frac{H\sum_{(u,v)\in E}\omega(u,v)\rho(u)\rho(v) - AB}{\sqrt{H\sum_{(u,v)\in E}\omega(u,v)\rho(u)^2 - A^2}\sqrt{H\sum_{(u,v)\in E}\omega(u,v)\rho(v)^2 - B^2}} \tag{7.59}$$

where $A = \sum_{(u,v)\in E}\omega(u,v)\rho(u)$ and $B = \sum_{(u,v)\in E}\omega(u,v)\rho(v)$ and $H = \sum_{e\in E}\omega(e)$ is the sum of all values of edges in E.

The above equation can also be written in terms of the entries of matrix W as:

$$r = \frac{H\sum_{i\neq j} w_{ij}\rho(v_i)\rho(v_j) - AB}{\sqrt{H\sum_{i\neq j} w_{ij}\rho(v_i)^2 - A^2}\sqrt{H\sum_{i\neq j} w_{ij}\rho(v_j)^2 - B^2}} \tag{7.60}$$

where $A = \sum_{i\neq j} w_{ij}\rho(v_i)$ and $B = \sum_{i\neq j} w_{ij}\rho(v_j)$, H is the sum of all values of matrix W, the indices i and j take values from 1 to n, $V = \{v_1,\ldots,v_n\}$ is the set of vertices, and $H = \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} w_{ij}$ is the sum of all values of matrix W.

Chapter 8

Analysis and Visualization of Greyscale Brain Functional Networks

8.1 BrainNetVis

BrainNetVis is an application, written in Java, that displays and analyzes greyscale networks from brain signals. The program runs as an application or it can be called as a function from Matlab. It implements a number of network indices and visualization techniques. We demonstrate it using EEG data from BCI IV competition and working at sensors' space. However the tool is not restricted to sensors' space since the input consists of synchronization matrices. Therefore our discussion is valid for generators' space as well.

BrainNetVis allows the user to define the location, label, font, color, shape and size of vertices. Vertices' centrality measures and attributes are visualized through the color of vertices via a colormap. On a second window one can see the the implemented centrality measures in a tabular format as well as a few network level statistics on a second tab of a tabbed pane (JTabbedPane). The user may select and delete vertices.

Each edge is visualized using shades of grey according to the rule $grey\ shade = 1-\omega(e)$. Alternatively, the edge values define the edge colors via a colormap (see Fig. 8.6 and 8.7). Another option is to convert a greyscale network $G = (V, E, \omega)$ to a graph $G' = (V, E')$ where $E' = \{e \in E \mid \omega(e) \geq \theta\}$, where θ is a user defined threshold and then to visualize graph G' [173]. The user may choose the width of edges to reflect the edge values. Higher values correspond to thicker lines. The direction of an edge is shown using an arrow that points to the destination vertex.

The graph visualization is based on two libraries created and maintained by Dimitris Andreou [15]. The first is called FlexiGraph and provides the basic graph functionality and the second provides basic graph visualization functionality and consists of a set of classes that are built on top of Piccolo 2D Graphics Framework [1]. The coordinates of the electrodes are determined by the stan-

[1] http://code.google.com/p/piccolo2d/

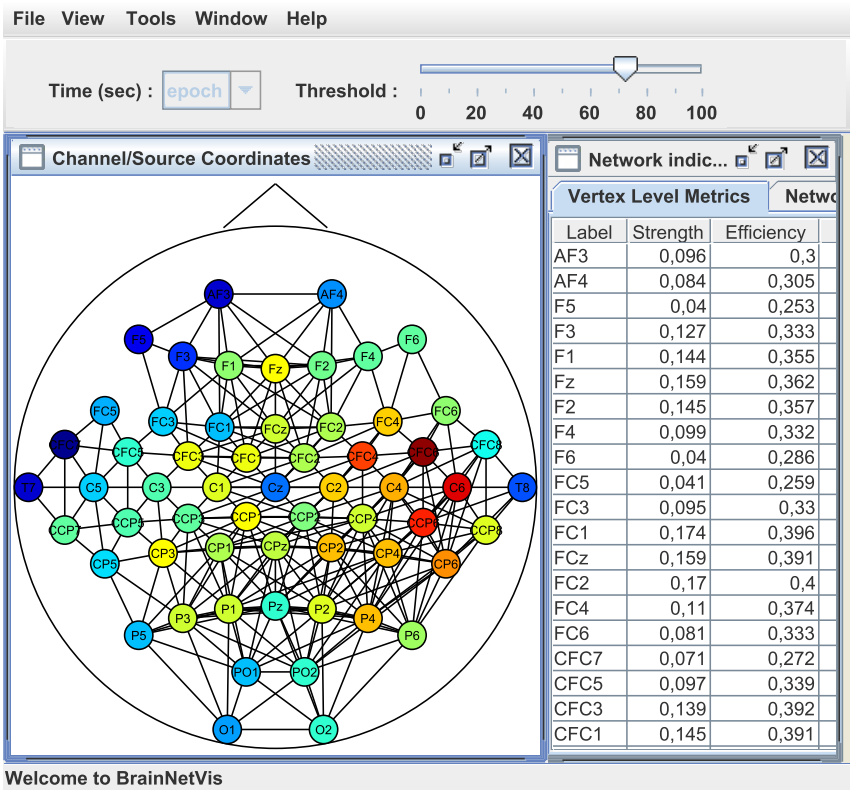


Figure 8.1: A view of BrainNetVis program.

dardized $10-5$ system for electrode placement [151].

The user may export the network views in a number of image format files including (encapsulated) postscript .eps and .ps, portable document format .pdf, window enhanced metafile .emf, bitmap .bmp, JPEG .jpg and many other formats.

In order to show the network structure, two network visualization techniques have been implemented. The stress majorization technique of Gansner et al. [77], the binary stress model of Koren et al. [115] (see Fig. 8.2) and the k-core decomposition of Alvarez-Hamelin et al. [13].

In a previous version [205] the program had an option to calculate brain topographic maps (see Fig. 8.4). However we excluded this option from the current version since the thin plate spline surface interpolation function [2], which is publicly available from koders[3], is written in C++ and is called from Java as a native method. However this restricts the portability of the program which is our main design goal. In the next version of the program we plan to translate the 3000 lines C++ thin plate spline surface interpolation code to Java and offer again the option to draw brain topographic maps. Additionally in the next version of BrainNetVis we plan a) to implement community structure (graph

[2] MgcInterp2DThinPlateSpline.cpp
[3] http://www.koders.com

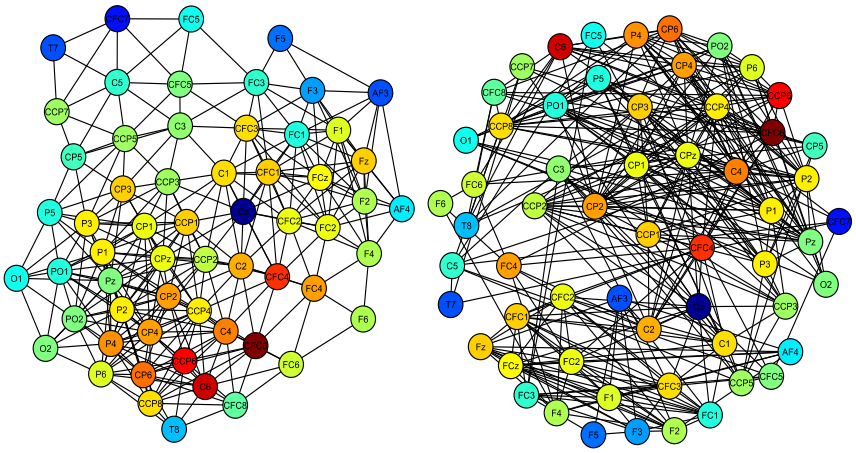


Figure 8.2: Visualization of network of Fig. 8.1 using multidimensional scaling (left) and binary stress (right).

Vertex Level Metrics | Network Level Metrics

Label	Strength	Efficiency	Betweenness (Lengths)	Betweenness (Values)	Hubbell	Eigenvalue	Subgraph	Entropy
Cz	0,401	0,569	0,021	0,025	0,688	0,148	17.587.985,593	0,076
CCP2	0,467	0,604	0,027	0,025	0,382	0,199	31.867.081,602	0,144
CPz	0,45	0,596	0,008	0,015	0,263	0,198	31.472.375,412	0,14
CCP1	0,469	0,604	0,031	0,029	0,211	0,195	30.630.309,23	0,139
CCP4	0,429	0,584	0,015	0,01	0,205	0,191	29.459.340,761	0,129
C1	0,413	0,573	0,033	0,041	0,171	0,143	16.417.348,016	0,071
CFC2	0,351	0,542	0,011	0,014	0,167	0,115	10.679.716,877	0,044
CFC1	0,36	0,546	0,021	0,018	0,15	0,115	10.595.468,92	0,044
F5	0,165	0,389	0,002	0,004	0,127	0,032	818.911,476	0,003
T8	0,234	0,439	0	0	0,113	0,106	9.010.966,369	0,034
CCP3	0,431	0,58	0,033	0,032	0,1	0,171	23.457.780,677	0,105
C4	0,397	0,559	0,007	0,011	0,089	0,175	24.504.187,587	0,106
Pz	0,4	0,567	0,002	0,005	0,081	0,182	26.510.054,724	0,113
CP2	0,444	0,593	0,01	0,011	0,074	0,198	31.464.196,479	0,138
C3	0,319	0,518	0,032	0,024	0,07	0,101	8.161.877,324	0,033
C2	0,459	0,595	0,045	0,041	0,046	0,184	27.309.679,396	0,124
CFC3	0,337	0,53	0,036	0,016	0,046	0,099	7.888.577,328	0,032
FC6	0,251	0,462	0,008	0,006	0,043	0,104	8.685.748,858	0,034
O1	0,23	0,437	0	0	0,035	0,107	9.182.138,803	0,035
FC1	0,305	0,507	0,003	0,008	0,022	0,08	5.183.560,003	0,02
CP1	0,443	0,592	0,029	0,02	0,019	0,189	28.841.797,341	0,128
FC3	0,285	0,484	0,019	0,015	0,008	0,07	3.960.186,11	0,016
O2	0,252	0,449	0	0	0,003	0,117	11.088.892,506	0,043

Figure 8.3: Vertex level (centrality) measures of network of Fig. 8.1.

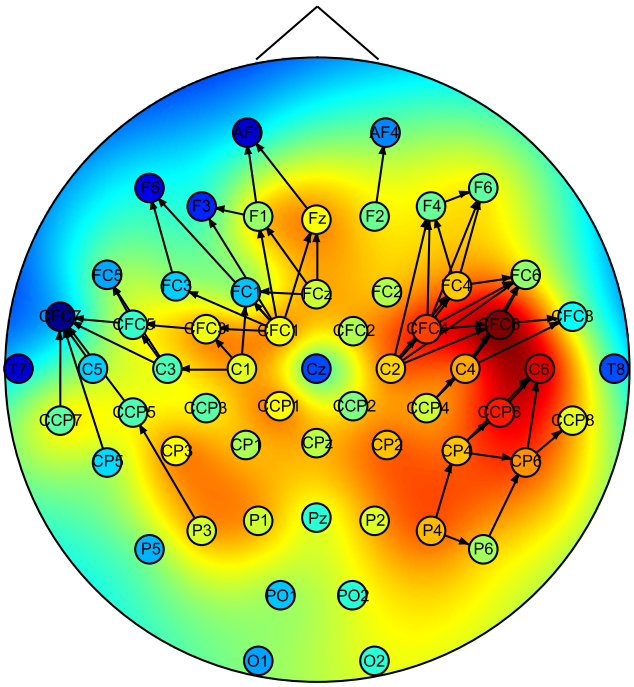


Figure 8.4: Spectral power density topographic map. See Fig. 8.9 for more details. The arrows correspond to edges with values above 0.52 of a greyscale network which was calculated using PDC.

clustering) algorithms as well as graph drawing algorithms that display the community structure and b) to extend its functionality to series of networks in order to study the dynamics of functional networks.

8.2 Case Study: Alcoholic and Control Subjects

8.2.1 Data acquisition and analysis

These EEG signals arise from 30 right-handed control subjects (had no personal or family history of any neurological disease, no age difference and normal vision or corrected normal vision) and 30 right-handed alcoholic subjects. The EEG signals were recorded in an electrically shielded, sound and light attenuated room. Participants sat in a reclined chair and fixated a point in the center of a computer display located 1m away from participants" eyes. Each subject was fitted with a 61-lead electrode cap (ECI, Electrocap International). All scalp electrodes were referred to Cz. Subjects were grounded with a nose electrode, and the electrode impedance was always below 5 kΩ. Two additional bipolar deviations were used to record the vertical and horizontal EOG. The signals were amplified with a gain of 10000 by Ep-A2 amplifiers (Sensorium, Inc., Charlotte, VT) with a bandpass between 0.02 and 50 Hz. The amplified signals were sampled at a rate of 256 Hz and ten artifact-free epochs of 1s were selected while

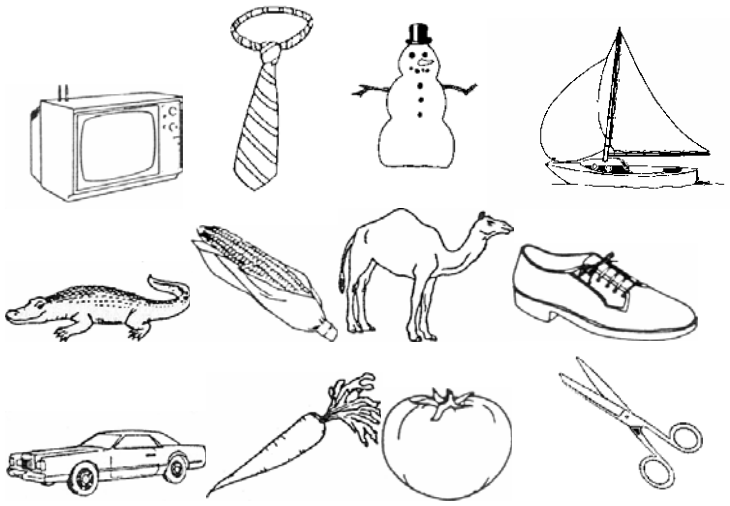


Figure 8.5: A few pictures from the standardized set of 260 pictures of Snodgrass and Vanderwart.

each subject performed a picture rehearsal task as described below. Trials with excessive eye and body movements (>73.3mV) were rejected on-line.

8.2.2 Experimental setting and test description

In this experiment, each subject was exposed to pictures of objects chosen from the 1980 Snodgrass and Vanderwart picture set [185] (see Fig. 8.5). These stimuli in each trial were randomized (but not repeated) and were presented on a white background for 300ms at the center of a computer monitor. Their size was approximately 5-10cm × 5-10cm, thus subtending a visual angle of 0.05^o-0.1^o. Ten trials were shown, with the interval between trials fixed to 3.2s. The participants were instructed to memorize the pictures in order to be able to identify them later.

8.2.3 Network construction

For each subject, trial and frequency band (0.5-4 Hz, 4-8 Hz, 8-13 Hz, 13-30 Hz, 30-45 Hz) the interdependence for each channel pair (there are $61 \cdot (61-1)/2$ channel pairs since the number of active EEG channels is 61) was calculated using the coherence and the RIM methods. The results were stored to 61x61 interdependence matrices W with elements ranging from 0 to 1.

8.2.4 Results

By visualizing, for each subject, the average per trial interdependence matrix we found that the alcoholic subjects have impaired synchronization of brain activity and loss of lateralization during the rehearsal process, most prominently in alpha band, as compared to control subjects (see Fig. 8.6).

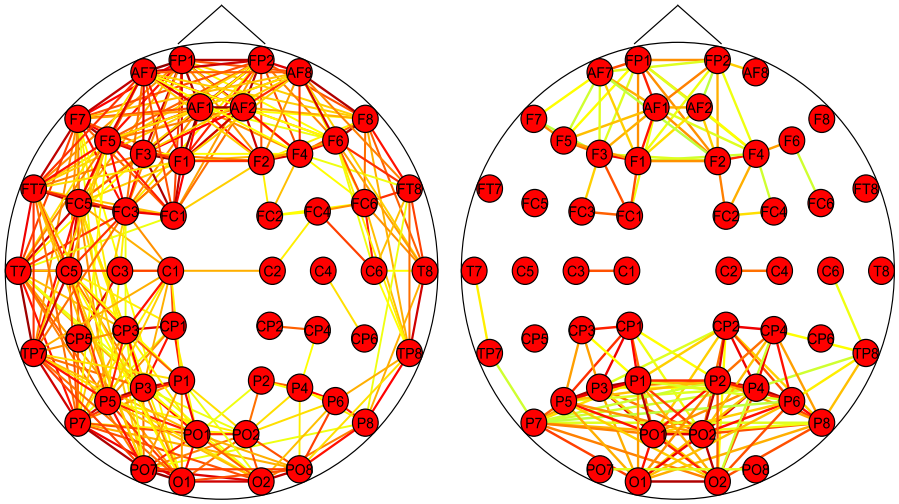


Figure 8.6: **Left:** Average network across trials of a control subject. **Right:** Average network across trials of an alcoholic subject. All networks were calculated at alpha band using mscoherence. Only edges with values above 0.59 are shown. Higher values are shown with red colors and wider edges. The channels Fpz, AFz, Fz, FCz, Cz, CPz, Pz, POz and Oz are not shown in this figure.

Lower beta synchronization was also evident in the alcoholic subject (Fig. 8.7). Similar findings were reported during moderate-to-heavy-alcohol intake during rest and mental rehearsal in the recent work of de Bruin et al. [48]. Older studies applied in hospitalized alcoholic patients, further support EEG coherence differentiation mostly in alpha and beta band [214]. Furthermore, the most prominent activation of the frontal and occipital brain regions is also justified, since they are thought to be involved during the picture recognition process [221]. Frontal areas are active during complex cognitive tasks, whereas reactivity in the occipital cortical regions is also found during fractal visual stimuli [170].

To measure the differentiation between the alcoholic and control patients we calculated network statistics for each subject and trial. Then we calculated the average and standard deviations across subjects and trials for each group: alcoholic and control. The networks were constructed using the mscoherence and RIM methods at beta band. The band was fixed to beta since we found [173, 171] that at this band there is maximum differentiation between the two groups. The clustering coefficient statistic shows a statistically significant difference between the two groups. Specifically control subjects' networks have higher clustering coefficient 8.1 than alcoholic subjects' networks. The assortativity statistic cannot differentiate between the two groups although the mean value of control group is higher than the mean value of alcoholic group. Both groups have networks that are assortative 8.2. Finally we found differentiation between the two groups using network efficiency 8.3. In tables 8.1 and 8.2 we can also see the effects of converting a greyscale network into a binary one through the thres_θ function. In table 8.3 three different functions are used to convert

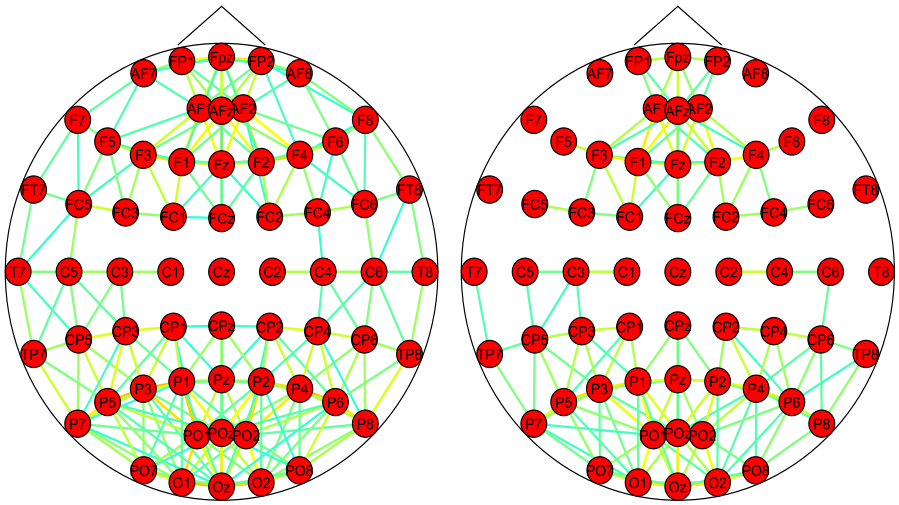


Figure 8.7: **Left:** Average network across control subjects and trials. **Right:** Average network across alcoholic subjects and trials. All networks were calculated at beta band using the RIM method. Only edges with values above 0.42 are shown. Higher values are shown with red colors and wider edges. In both figures we see that due to network averaging the strongest edges connect geographically neighboring channels.

edge strength into edge length.

8.3 Case Study: Motor Imagery

A brain-computer interface (BCI) system provides direct communication between the brain and a computer or a device. A BCI can transform mental decision and reaction into control commands by analyzing the bioelectrical brain activity. Currently, BCI systems ignore the synchronization between EEG signals detected from different recording sites and very few studies have addressed this issue for improving the performance of BCI systems [98]. In our work [204] we examined whether there are differences between the greyscale networks derived from EEG signals during different mental tasks of a subject. We found that at sensors' space it is hard to distinguish between two motor imagery states (namely left hand and foot) using the corresponding greyscale networks because these networks are dominated by irrelevant edges. However we managed to find differences between the average left hand and foot greyscale networks by comparing them with the average idle state greyscale network.

8.3.1 Experimental Setup

We used data sets 1 provided by the Berlin BCI group for BCI competition IV [4]. These data sets were recorded from four healthy subjects. Brain activity was

[4]http://ida.first.fraunhofer.de/projects/bci/competition_iv/

Table 8.1: Clustering coefficient statistics. The networks were calculated using the RIM method 6.5.3 at beta frequency band. Clustering coefficient was calculated for each trial of each subject of alcoholic or control group. Then the mean and standard deviations were calculated for each group. The last line of the table has "yes" when there is statistically important differentiation between control and alcoholic clustering coefficient and "no" otherwise.

Network type		Greyscale	Binary
Alcoholic (450 samples)	mean	0.3058	0.5278
	std	0.0628	0.0768
Control (463 samples)	mean	0.37825	0.6142
	std	0.0550	0.0758
Differentiation ($\alpha = 0.05$)		yes	yes

Table 8.2: Assortativity statistics. The statistics were calculated as in table 8.1.

Network type		Greyscale	Binary
Alcoholic (450 samples)	mean	0.3260	0.3979
	std	0.1472	0.1930
Control (463 samples)	mean	0.34062	0.4169
	std	0.1498	0.1808
Differentiation ($\alpha = 0.05$)		no	no

Table 8.3: Network efficiency statistics. The statistics were calculated as in table 8.1. Additionally in the calculations we take into account the effect of the non-increasing function used to transform edge strength into edge length. Three functions were considered: the inverse function, the $\log_2$ function and the inverse threshold function that assigns length 1 to an edge and length $+\infty$ to absence of an edge between two vertices. The inverse threshold function is the usual graph theoretic distance function applied to graph constructed using the threshold function $\text{thres}_\theta(x)$, where $\theta = 0.4$.

Edge value to length function		g(x)=$\frac{1}{x}$	g(x)=$1 - log_2(x)$	g(x)=$\begin{cases} +\infty & \text{if } x < \theta \\ 1 & \text{if } x \geq \theta \end{cases}$
Alcoholic	mean	0.2258	0.3179	0.3125
	std	0.0489	0.0455	0.1017
Control	mean	0.2611	0.3684	0.4241
	std	0.0457	0.0436	0.1027
Differentiation ($\alpha = 0.05$)		yes	yes	yes

recorded with multi-channel EEG amplifiers (Brain Products GmbH, Germany) using 59 channels band-pass filtered between 0.05 and 200 Hz and sampled at 1000 Hz. Subjects sat in a comfortable chair with arms placed on armrests. All recordings for one individual subject were recorded on the same day. In the whole session motor imagery was performed without feedback. For each subject two classes of motor imagery were selected: left hand (L), and foot (F) (side chosen by the subject; optionally also both feet). Arrows pointing left or down were presented on a computer screen as visual cues for left hand and foot respectively. Cues were displayed for a period of 4s during which the subject was instructed to perform the cued motor imagery task. These periods were interleaved with 2s of blank screen and 2s with a fixation cross shown in the center of the screen. The fixation cross was superimposed on the cues, i.e. it was shown for 6s. For each subject both training and test data sets are provided. In our study we use only the training data sets since they are accompanied with complete marker information. Additional information about the experimental setup exists in paper [34].

Power Spectral Density Estimation

The multichannel EEG was filtered using the Common Spatial Patterns (CSP) spatial filter and then the Power Spectral Density (PSD) was calculated using Welch's method.

PDC Estimation

Due to limited number of data samples, PDC 6.12 was calculated between each pair of channels by taking into account and removing the linear effect of O1 and O2 channels [136].

8.3.2 Results

For each subject, frequency band and synchronization method we calculated synchronization matrices for every left (L), foot (F) or idle (I) 4 sec period, within the subject's session, that passed the visual check for artifacts. Specifically we calculated more than 80 synchronization matrices for each state L, F and I and for each subject. Then for each subject we calculated the average matrix per state in order to find differences in channels' synchronization between the states. In the following we call L, F and I the average left, foot and idle state networks. The first observation is that functional networks constructed from motor imagery EEG data are dominated by edges that are irrelevant to the motor imagery tasks. Especially close-by electrodes are highly synchronized which reflects redundancies in the measurement due to volume conduction rather than brain interaction [101] (see Fig. 8.8). The second observation is that networks L and F look and are very similar. Network L has clustering and assortativity coefficients 0.51 and 0.126 respectively, while network F has clustering and assortativity coefficients 0.49 and 0.13 respectively. These results agree with the results of Calmels et al. [55]. To extract information from networks L and F we compare each one of them with network I. We consider a network that has the same set of vertices as networks L, F and I and has an edge from vertex j to vertex i ($i \leftarrow j$ due to convention of PDC) if and only if $W_{ij}^{(L)} - W_{ij}^{(I)} > 0$ (resp.

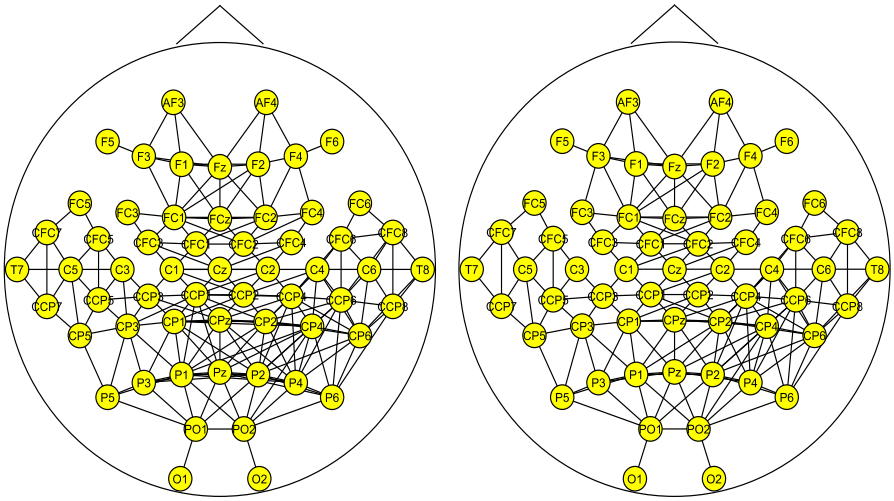
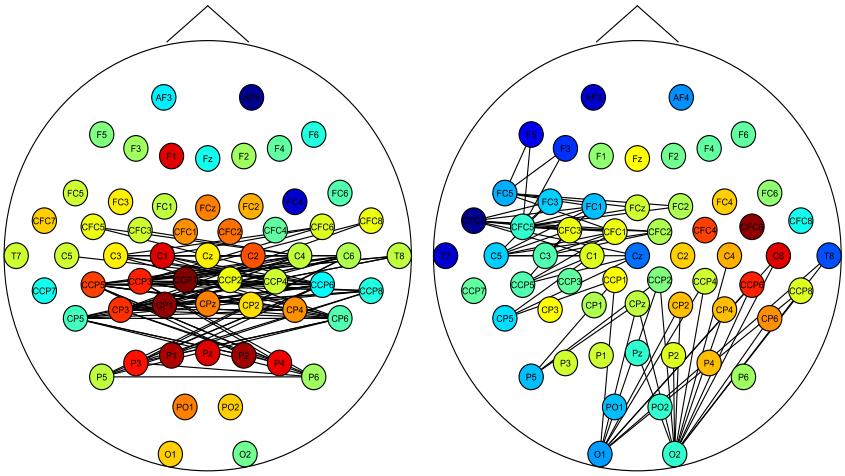
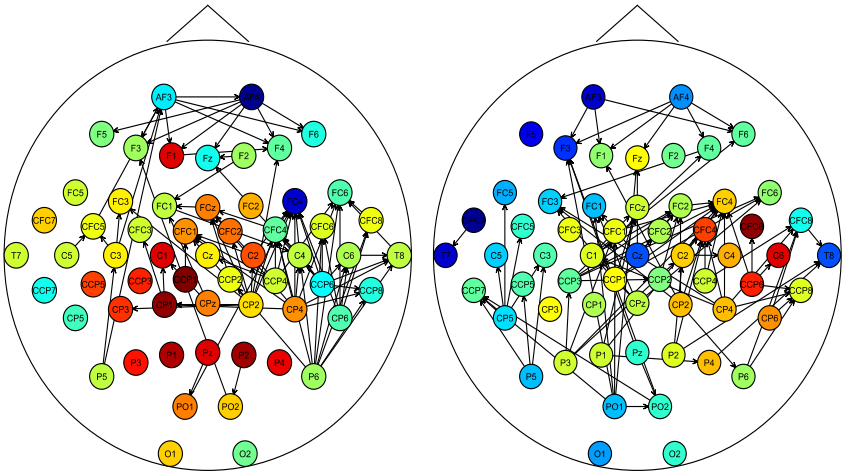


Figure 8.8: L (left figure) and F (right figure) networks at mu band for subject a. The networks were calculated using the nonlinear RIM measure. Only edges with values above 0.8 are shown.

$W_{ij}^{(F)} - W_{ij}^{(I)} > 0$). The weight of edge (j, i) of network L minus I (resp. F minus I) is set to $W_{ij}^{(L)} - W_{ij}^{(I)}$ (resp. $W_{ij}^{(F)} - W_{ij}^{(I)}$). By subtracting network I from network L (resp. F) most noisy edges disappear and in the resulting network one can discern edges related to motor imagery task. For subject a, when networks have been calculated using PDC then the resulting network L minus I has long edges that span both hemispheres and which can be attributed to motor imagery task [136] or to occipital/parietal alpha activity [16, 70]. However this is not apparent in network F minus I. When networks have been calculated using the RIM or the magnitude square coherence methods the long edges between hemispheres of network L minus I are present but mixed with other edges that are adjacent to vertices O1, O2, AF3 and AF4 (see Fig. 8.9). We repeated this procedure by changing the role of networks I and L (resp. I and F). Namely from network I we subtracted network L (resp. F). In the left (resp. right) graph of Fig. 8.9 we see that network I minus L (resp. I minus F) has more edges close to the left hand (resp. foot) area of the sensorimotor cortex. The above results are valid for subject a. There is inter-subject variability since for subjects b and f the long edges across hemispheres appear in I minus L and in I minus F networks. Finally, from the tables of Fig. 8.9) we see that the important, according to Hubbell's centrality, vertices of I minus L and I minus F networks for subject a are relevant to state L (resp. F).



(a)



(b)

Vertex Level Metrics | Network Level

Label	Strength	Efficiency	Hubbell
FC4	1,467	1,027	1,653
CCP6	1,842	1,778	1,169
CFC4	1,214	1,076	1,053
C4	1,501	1,384	1,02

Vertex Level Metrics | Network Level

Label	Strength	Efficiency	Hubbell
Cz	0,742	1,131	1,031
C2	1,143	1,219	0,902
C1	0,916	1,112	0,895
CCP3	0,865	1,283	0,881

(c)

Figure 8.9: Subject a and mu band. **a)** Network L minus I (left figure) [resp. F minus I (right figure)]. The networks were calculated using the RIM measure. The color of a vertex shows the (average) PSD of L (resp. F) state. Red colors correspond to higher values than blue colors. **b)** Network I minus L (left figure) [resp. I minus F (right figure). The networks were calculated using PDC. **c)** Ranking of vertices of network I minus L (resp. I minus F) according to Hubbell's centrality index.

Chapter 9

Conclusions & Future Work

In our study we considered two characteristic examples of biomedical networks: Gene Ontology and brain functional networks. In order to visualize GO using space filling techniques we generalized treemaps to DAGmaps [207]. DAGmaps impose adjacency constraints on the rectangles and depending on its structure a DAG may or may not admit a DAGmap. We proved that deciding whether or not a DAG admits a DAGmap is NP-complete. Then we studied two special cases of the DAGMAP problem and a vertex duplication and drawing heuristic algorithm. In the first special case we restricted the input of DAGMAP problem to the class of TTSP digraphs. We showed that any TTSP digraph admits a DAGmap which can be drawn in linear time. Drawing TTSP digraphs as DAGmaps is a proper generalization of treemaps since in both cases the layout function that partitions a rectangle into subrectangles does not depend on the structure of the DAG. Therefore we can choose a layout function with desirable properties such as one that reduces the aspect ratios of the rectangles. In the second special case we restricted the ways of partitioning the rectangles. Namely the rectangles are sliced in only one direction (e.g., in the vertical direction). We call this special case of the problem ONE-DIMENSIONAL DAGMAP. We proved that a DAG admits a one-dimensional DAGmap if and only if it admits a directed ε-visibility representation. Therefore we established a one-to-one connection between two important and previously seemingly unrelated types of graph drawing: one-dimensional DAGmaps on the one hand, and on the other hand directed ε-visibility representations. Based on this connection we designed linear time testing and drawing algorithms for the ONE-DIMENSIONAL DAGMAP problem. The heuristic algorithm that we propose decomposes a DAG into component st-graphs using dominance relationships and then combines the drawings of the component st-graphs. In case that an st-graph does not admit a DAGmap then it is transformed into a TTSP digraph via vertex duplications. This algorithm performs fewer vertex duplications than the transformation of a DAG into a tree when the DAG can be decomposed into component st-graphs. Finally we presented a program which is called DAGmap View. This program implements the novel feature of separate layout and nesting functions and is an ideal tool to visualize and navigate through GO.

For the analysis and visualization of brain functional networks we created a program which is called BrainNetVis. These networks have values associated with their edges and can be visualized with different shades of grey. For this

reason we call them greyscale networks. In order to analyze greyscale networks we use network measures that take into account the edge values. Most of these measures were selected from the literature. However we also contributed to the development of the theory of greyscale networks by proposing: a) a function that converts edge strength into edge lengths; b) a formula that calculates the importance of a vertex due to its position on a network as well as due to its attributes and c) some generalizations of graph theoretic measures to greyscale networks. We demonstrate BrainNetVis using EEG data from two different studies; a) a study that examine EEG correlates of genetic predisposition to alcoholism and b) a study that examine EEG correlates to motor imagery tasks. In the case of alcoholism, we found differentiation between alcoholic and control subjects at beta band. In the case of motor imagery tasks, we found that it is hard to differentiate between two imagery states (left hand and foot) by comparing the functional connectivity networks at sensors' space. This difficulty arises from the fact that the two networks differ only in some weak edges while they have the same strong edges. In order to clean the strong edges, which are irrelevant to motor imagery tasks, we proposed a heuristic method that subtracts the idle state network from the left hand (resp. foot) network.

In our analysis of greyscale networks we considered either local measures (at the level of vertices) or global measures (at the level of the whole network). The next step is to consider graph clustering (also known as community structure) and of course to implement graph clustering in BrainNetVis. Graph clustering is important since one property observed in many networks is the existence of a modular (or cluster) structure. We recall here that a network is modular if it exhibits significantly more intramodular connections compared to intermodular connections. Especially in brain functional networks, some cortical regions are supposed to share a large number of functional relationships during the performance of several motor and cognitive tasks. This characteristic leads to the formation of highly connected clusters within a brain network. These functional groups consist of a certain number of different cerebral areas that are cooperating more intensively in order to complete a task successfully [134, 135, 184].

Another important extension of BrainNetVis is to visualize and analyze dynamic networks, where a dynamic network is regarded as a sequence of networks with different timestamps (or temporal intervals) [111, 110]. Dynamic networks, which are a hot research topic, capture the dynamic nature of brain functional networks.

A visualization of brain networks can combine both space filling and node-link techniques. The paper of Holten [92] on visualization of adjacency relations in hierarchical data is an excellent guide of how one can do this. The hierarchical relations could be anatomical divisions and subdivisions of the brain and the adjacency relations denote connectivity among brain regions of the same granularity. The hierarchical relations can be visualized with space filling techniques while the adjacency relations with node-link techniques.

As far as the connection between DAGmaps and three dimensional ε-visibility representations is concerned it remains to be investigated under what restrictions the two problems are equivalent. This may yield wonderful theoretical results as well as elegant heuristic algorithms that transform a DAG G into a DAG G' that admits a DAGmap. Last but not least in my future plans is the improvement of the DAGFmap View program by taking into account some suggestions from biologists, and then the release of the program to the public.

Bibliography

[1] Achard, S., Bullmore, E.: Efficiency and cost of economical brain functional networks. PLoS Computational Biology 3(2), e17 (2007)

[2] Ahnert, S.E., Garlaschelli, D., Fink, T.M.A., Caldarelli, G.: Ensemble approach to the analysis of weighted networks. Physical Review E 76(1), 016101 (2007)

[3] Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1972)

[4] Albert, R., Barabási, A.L.: Emergence of scaling in random networks. Rev. Mod. Phys. 74(1), 47–97 (2002)

[5] Albertson, D.G., Thomson, J.N.: The pharynx of caenorhabditis elegans. Philosophical Transactions of the Royal Society of London - Series B 275(938), 299–325 (1976)

[6] Allen, F.E., Cocke, J.: Graph theoretic constructs for program control flow analysis. Tech. rep., IBM Res. Rep. RC 3923, IBM T.J. Watson Research Center (1972)

[7] Allen, F.E.: Control flow analysis. SIGPLAN Notices 5(7), 1–19 (1970)

[8] Allesina, S., Bodini, A.: Who dominates whom in the ecosystem? Energy flow bottlenecks and cascading extinctions. Journal of Theoretical Biology 230(3), 351–358 (2004)

[9] Allesina, S., Bodini, A., Bondavalli, C.: Secondary extinctions in ecological networks: Bottlenecks unveiled. Ecological Modelling 194(1-3), 150–161 (2006)

[10] Allesina, S., Pascual, M.: Googling food webs: Can an eigenvector measure species' importance for coextinctions? PLoS Computational Biology 5(9), e1000494 (2009)

[11] Alon, U.: Network motifs: theory and experimental approaches. Nature Reviews Genetics 8(6), 450–461 (2007)

[12] Alstrup, S., Harel, D., Lauridsen, P.W., Thorup, M.: Dominators in linear time. SIAM Journal on Computing 28(6), 2117–2132 (1999)

[13] Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the k-core decomposition. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems 18, pp. 41–50. MIT Press, Cambridge, MA (2006)

[14] Amyeen, M., Fuchs, W., Pomeranz, I., Boppana, V.: Fault equivalence identification using redundancy information and static and dynamic extraction. In: 19th IEEE VLSI Test Symposium. pp. 124–130 (2001)

[15] Andreou, D.: Flexigraph, URL `http://www.csd.uoc.gr/~andreou/flexigraph.html`

[16] Andrew, C., Pfurtscheller, G.: Dependence of coherence measurements on EEG derivation type. Medical and Biological Engineering and Computing 34(3), 232–238 (1996)

[17] Andrzejak, R.G., Kraskov, A., Stögbauer, H., Mormann, F., Kreuz, T.: Bivariate surrogate techniques: Necessity, strengths, and caveats. Physical Review E 68(6), 066202 (2003)

[18] Arnhold, J., Grassberger, P., Lehnertz, K., Elger, C.E.: A robust method for detecting interdependences: application to intracranially recorded EEG. Physica D 134(4), 419–430 (1999)

[19] Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium. Nature Genetics 25(1), 25–29 (2000), URL `http://www.geneontology.org/`

[20] Baccalá, L.A., Sameshima, K.: Partial directed coherence: a new concept in neural structure determination. Biological Cybernetics 84(6), 463–474 (2001)

[21] Baccalá, L.A., Sameshima, K.: Generalized partial directed coherence. In: 15th International Conference on Digital Signal Processing. pp. 163–166. IEEE Computer Society (2007)

[22] Badii, R., Broggi, G., Derighetti, B., Ravani, M.: Dimension increase in filtered chaotic signals. Physical Review Letters 60(11), 979–982 (1988)

[23] Baehrecke, E.H., Dang, N., Babaria, K., Shneiderman, B.: Visualization and analysis of microarray and gene ontology data with treemaps. BMC Bioinformatics 5(84) (2004)

[24] Barrat, A., Barthélemy, M., Pastor-Satorras, R., Vespignani, A.: The architecture of complex weighted networks. Proceedings of the National Academy of Sciences USA 101(11), 3747–3752 (2004)

[25] Barrat, A., Weigt, M.: On the properties of small-world network models. The European Physical Journal B 13(3), 547–560 (2000)

[26] Başar, E.: The theory of the whole-brain-work. International Journal of Psychophysiology 60(2), 133–138 (2006)

[27] Bassett, D.S., Meyer-Lindenberg, A., Achard, S., Duke, T., Bullmore, E.: Adaptive reconfiguration of fractal small-world human brain functional networks. Proceedings of the National Academy of Sciences USA 103(51), 19518–19523 (2006)

[28] Beauchamp, M.A.: An improved index of centrality. Behavioral Science 10(2), 161–163 (1965)

[29] Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Partitioning a square into rectangles: NP-completeness and approximation algorithms. Algorithmica 34(3), 217–239 (2002)

[30] Bederson, B.B., Shneiderman, B., Wattenberg, M.: Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. ACM Transactions on Graphics 21(4), 833–854 (2002)

[31] Bennett, M., Schatz, M.F., Rockwood, H., Wiesenfeld, K.: Huygens's clocks. Proceedings of the Royal Society A 458(2019), 511–762 (2002)

[32] Berriz, G.F., White, J.V., King, O.D., Roth, F.P.: Gofish finds genes with combinations of gene ontology attributes. Bioinformatics 19(6), 788–789 (2003), URL `http://llama.med.harvard.edu/gofish/GoFishWelcome.html`

[33] Biggs, N.: Algebraic Graph Theory. Cambridge University Press (1993)

[34] Blankertz, B., Dornhege, G., Krauledat, M., Müller, K.R., Curio, G.: The non-invasive berlin brain-computer interface: Fast acquisition of effective performance in untrained subjects. NeuroImage 37(2), 539–550 (2007)

[35] Blinowska, K.J., Malinowski, M.: Non-linear and linear forecasting of the EEG time series. Biological Cybernetics 66(2), 159–165 (1991)

[36] Bodini, A., Bellingeri, M., Allesina, S., Bondavalli, C.: Using food web dominator trees to catch secondary extinctions in action. Phil. Trans. R. Soc. B 364(1524), 1725–1731 (2009)

[37] Bollobás, B., Riordan, O.M.: Mathematical results on scale-free random graphs. In: Bornholdt, S., Schuster, H.G. (eds.) Handbook of Graphs and Networks, pp. 1–34. Wiley InterScience (2003)

[38] Bonacich, P.: Factoring and weighting approaches to status scores and clique identification. Journal of Mathematical Sociology 2(1), 113–120 (1972)

[39] Bonacich, P.: Power and centrality: A family of measures. The American Journal of Sociology 92(5), 1170–1182 (1987)

[40] Booth, T.E.: Computing the higher k-eigenfunctions by monte carlo power iteration: A conjecture. International Journal of Nonlinear Science 143(3), 291–300 (2003)

[41] Borgatti, S.P.: Centrality and network flow. Social Networks 27(1), 55–71 (2005)

[42] Bose, P., Everett, H., Fekete, S.P., Houle, M.E., Lubiw, A., Meijer, H., Romanik, K., Rote, G., Shermer, T.C., Whitesides, S., Zelle, C.: A visibility representation for graphs in three dimensions. Journal of Graph Algorithms and Applications 2(3), 1–16 (1998)

[43] Brandes, U.: A faster algorithm for betweenness centrality. Journal of Mathematical Sociology 25(2), 163–177 (2001)

[44] Brandes, U.: On variants of shortest-path betweenness centrality and their generic computation. Social Networks 30(2), 136–145 (2008)

[45] Brandes, U., Cornelsen, S.: Phylogenetic graph models beyond trees. Discrete Applied Mathematics 157(10), 2361–2369 (2009)

[46] Bressler, S.L., Tognoli, E.: Operational principles of neurocognitive networks. International Journal of Psychophysiology 60(2), 139–148 (2006)

[47] Brillinger, D.R.: Time Series: Data Analysis and Theory. Classics in Applied Mathematics, SIAM (2001)

[48] de Bruin, E.A., Stam, C.J., Bijl, S., Verbaten, M.N., Kenemans, J.L.: Moderate-to-heavy alcohol intake is associated with differences in synchronization of brain activity during rest and mental rehearsal. International Journal of Psychophysiology 60(3), 304–314 (2006)

[49] Bruls, M., Huizing, K., van Wijk, J.: Squarified treemaps. In: de Leeuw, W., van Liere, R. (eds.) Data Visualization 2000, Proceedings of the Second Joint Eurographics and IEEE TCVG Symposium on Visualization. pp. 33–42 (2000)

[50] Buchsbaum, A.L., Kaplan, H., Rogers, A., Westbrook, J.R.: A new, simpler linear-time dominators algorithm. ACM Trans. Program. Lang. Syst. 20(6), 1265–1296 (1998)

[51] Buchsbaum, A.L., Kaplan, H., Rogers, A., Westbrook, J.R.: Corrigendum: a new, simpler linear-time dominators algorithm. ACM Trans. Program. Lang. Syst. 27(3), 383–387 (2005)

[52] Bullmore, E., Sporns, O.: Complex brain networks: graph theoretical analysis of structural and functional systems. Nature Reviews Neuroscience 10(3), 186–198 (2009)

[53] Burt, R.S.: Positions in networks. Social Forces 55(1), 93–122 (1976)

[54] Burt, R.S.: Structure Version 4.2 Reference Manual. Center for the Social Sciences, Columbia University (1991)

[55] Calmels, C., Hars, M., Holmes, P., Jarry, G., Stam, C.J.: Non-linear EEG synchronization during observation and execution of simple and complex sequential finger movements. Experimental Brain Research 190(4), 389–400 (2008)

[56] Cancer Genome Anatomy Project: CGAP GO browser, URL `http://cgap.nci.nih.gov/Genes/GOBrowser`

[57] Cao, L.: Practical method for determining the minimum embedding dimension of a scalar time series. Physica D 110(1-2), 43–50 (1997)

[58] Cooper, K.D., Harvey, T.J., Kennedy, K.: A simple, fast dominance algorithm (2001), available on line at http://www.cs.rice.edu/∼keith/EMBED/dom.pdf

[59] David, O., Cosmelli, D., Friston, K.J.: Evaluation of different measures of functional connectivity using a neural mass model. NeuroImage 21(2), 659–673 (2004)

[60] Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Upper Saddle River, NJ (1998)

[61] Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theoretical Computer Science 61(2-3), 175–198 (1988)

[62] Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica 15(4), 302–318 (1996)

[63] Ding, M., Bressler, S.L., Yang, W., Liang, H.: Short-window spectral analysis of cortical event-related potentials by adaptive multivariate autoregressive modeling: data preprocessing, model validation, and variability assessment. Biological Cybernetics 83(1), 35–45 (2000)

[64] Duarte, N.C., Becker, S.A., Jamshidi, N., Thiele, I., Mo, M.L., Vo, T.D., Srivas, R., Palsson, B.: Global reconstruction of the human metabolic network based on genomic and bibliomic data. Physical Review Letters 104(6), 1777–1782 (2007)

[65] Dufour, J.M., Renault, E.: Short run and long run causality in time series: theory. Econometrica 66(5), 1099–1125 (1998)

[66] Durek, P., Walther, D.: The integrated analysis of metabolic and protein interaction networks reveals novel molecular organizing principles. BMC System Biology 2(100) (2008)

[67] Eades, P., Whitesides, S.: Drawing graphs in two layers. Theoretical Computer Science 131(2), 361–374 (1994)

[68] Eichler, M.: On the evaluation of information flow in multivariate systems by the directed transfer function. Biological Cybernetics 94(6), 469–482 (2006)

[69] Estrada, E., Rodríguez-Velázquez, J.A.: Subgraph centrality in complex networks. Physical Review E 71(5), 056103 (2005)

[70] Florian, G., Andrew, C., Pfurtscheller, G.: Do changes in coherence always reflect changes in functional coupling? Electroenceph. Clin. Neurophysiol. 106(1), 87–91 (1998)

[71] Fox, M.D., Raichle, M.E.: Spontaneous fluctuations in brain activity observed with functional magnetic resonance imaging. Nature Reviews Neuroscience 8(9), 700–711 (2007)

[72] Freeman, L.C.: A set of measures of centrality based upon betweenness. Sociometry 40(1), 35–41 (1977)

[73] Freeman, L.C.: Centrality in social networks conceptual clarification. Social Networks 1(3), 211–292 (1979)

[74] Freeman, L.C., Borgatti, S.P., White, D.R.: Centrality in valued graphs: A measure of betweenness based on network flow. Social Networks 13(2), 105–204 (1991)

[75] Friston, K.J.: Functional and effective connectivity in neuroimaging: A synthesis. Human Brain Mapping 2(1-2), 56–78 (1994)

[76] Committee on Network Science for Future Army Applications, N.R.C.: Network Science. National Academic Press (2005)

[77] Gansner, E.R., Koren, Y., North, S.: Graph drawing by stress majorization. In: GD 2004. LNCS, vol. 3383, pp. 239–250. Springer-Verlag (2005)

[78] Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing 4(4), 397–411 (1975)

[79] The Gene Ontology Consortium: AmiGO, URL `http://amigo.geneontology.org/cgi-bin/amigo/go.cgi`

[80] The Gene Ontology Consortium: OBO-Edit, URL `http://www.oboedit.org/`

[81] Georgiadis, L., Tarjan, R.E.: Finding dominators revisited: extended abstract. In: ACM-SIAM SODA '04. pp. 869–878. SIAM (2004)

[82] Georgiadis, L., Tarjan, R.E., Werneck, R.F.: Finding dominators in practice. Journal of Graph Algorithms and Applications 10(1), 69–94 (2006)

[83] Godsil, C., Royle, G.: Algebraic Graph Theory, Graduate Texts in Mathematics, vol. 207. Springer (2001)

[84] Granger, C.W.J.: Investigating causal relations by econometric models and cross-spectral methods. Econometrica 37(3), 424–438 (1969)

[85] Gubernatis, J., Booth, T.E.: Multiple extremal eigenpairs by the power method. Journal of Computational Physics 227(19), 8508–8522 (2008)

[86] Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: GD2000. LNCS, vol. 1984, pp. 77–90 (2001)

[87] Hagmann, P., Cammoun, L., Gigandet, X., Meuli, R., Honey, C.J., Wedeen, V.J., Sporns, O.: Mapping the structural core of human cerebral cortex. PLoS Biology 6(7), e159 (2008)

[88] Harel, D.: A linear algorithm for finding dominators in flow graphs and related problems. In: ACM STOC '85. pp. 185–194. ACM (1985)

[89] Higham, D.J., Rašajskim, M., Pržulj, N.: Fitting a geometric graph to a protein-protein interaction network. Bioinformatics 24(8), 1093–1099 (2008)

[90] Hoffmann, S., Falkenstein, M.: The correction of eye blink artefacts in the EEG: A comparison of two prominent methods. PLoS ONE 3(8), e3004 (2008)

[91] Holme, P., Park, S.M., Kim, B.J., Edling, C.R.: Korean university life in a network perspective: Dynamics of a large affiliation network. Physica A 373(1), 821–830 (2007)

[92] Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Transactions on Visualization and Computer Graphics 12(5), 741–748 (2006)

[93] Honey, C.J., Kötter, R., Breakspear, M., Sporns, O.: Network structure of cerebral cortex shapes functional connectivity on multiple time scales. Proceedings of the National Academy of Sciences USA 104(24), 10240–10245 (2007)

[94] Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM Journal on Computing 2(3), 135–158 (1973)

[95] Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press (1990)

[96] Horwitz, B.: The elusive concept of brain connectivity. NeuroImage 19(2), 466–470 (2003)

[97] Hsiao, C.: Autoregressive modeling and causal ordering of economic variables. Journal of Economic Dynamics and Control 4, 243–259 (1982)

[98] Hu, J., Mu, Z., Wang, J.: Phase locking analysis of motor imagery in brain-computer interface. In: BMEI 2008. vol. 2, pp. 478–481. IEEE Computer Society (2008)

[99] Huang, X., Lai, W.: On the structural algorithm of filtering graphs for layout. In: VIP '05. pp. 33–42. Australian Computer Society, Inc. (2004)

[100] Hubbell, C.H.: An input-output approach to clique identification. Sociometry 28, 377–399 (1965)

[101] Ioannides, A.A.: Dynamic functional connectivity. Current Opinion in Neurobiology 17(2), 161–170 (2007)

[102] Johnson, B., Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the IEEE Visualization "91. pp. 284–291 (1991)

[103] Jordán, F., Benedek, Z., Podani, J.: Quantifying positional importance in food webs: A comparison of centrality indices. Ecological Modelling 205(1-2), 270–275 (2007)

[104] Jünger, M., Leipert, S.: Level planar embedding in linear time. Journal of Graph Algorithms and Applications 6(1), 67–113 (2002)

[105] Kamiński, M., Blinowska, K.J.: A new method of the description of the information flow in the brain structures. Biological Cybernetics 65(3), 203–210 (1991)

[106] Kamiński, M., Ding, M., Truccolo, W.A., Bressler, S.L.: Evaluating causal relations in neural systems: Granger causality, directed transfer function and statistical assessment of significance. Biological Cybernetics 85(2), 145–157 (2001)

[107] Kant, G., Liotta, G., Tamassia, R., Tollis, I.G.: A visibility representation for graphs in three dimensions. Area requirement of visibility representations of trees 62(2), 81–88 (1997)

[108] Karlebach, G., Shamir, R.: Modelling and analysis of gene regulatory networks. Nature Reviews Molecular Cell Biology 9(10), 770 (2008)

[109] Katz, L.: A new status index derived from sociometric analysis. Psychometrika 18(1), 39–43 (1953)

[110] Kim, M.S., Han, J.: Chronicle: A two-stage density-based clustering algorithm for dynamic networks. In: Discovery Science. pp. 152–167 (2009)

[111] Kim, M.S., Han, J.: A particle-and-density based evolutionary clustering method for dynamic networks. In: PVLDB. vol. 2, pp. 622–633 (2009)

[112] Klir, G.J., Folger, T.A.: Fuzzy Sets, Uncertainty and Information. Prentice Hall (1988)

[113] Kloepper, T.H., Huson, D.H.: Drawing explicit phylogenetic networks and their integration into splitstree. BMC Evolutionary Biology 8(1), 22 (2008)

[114] Koles, Z.: The quantitative extraction and topographic mapping of the abnormal components in the clinical eeg. Electroencephalography and Clinical Neurophysiology 79(6), 440–447 (1991)

[115] Koren, Y., Çivril, A.: The binary stress model for graph drawing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 193–205. Springer-Verlag (2009)

[116] Kranakis, E., Krizanc, D., Urrutia, J.: On the number of directions in visibility representations of graphs. In: GD '94. LNCS, vol. 894, pp. 167–176. Springer-Verlag (1995)

[117] Kugiumtzis, D.: Surrogate data test on time series. In: Modelling and Forecasting Financial Data Techniques of Nonlinear Dynamics, Studies in Computational Finance, vol. 2. Springer (2002)

[118] Latora, V., Marchiori, M.: Efficient behavior of small-world networks. Physical Review Letters 87(19), 198701 (2001)

[119] Latora, V., Marchiori, M.: Economic small-world behavior in weighted networks. The European Physical Journal B 32(2), 249–263 (2003)

[120] Lengauer, T., Tarjan, R.E.: A fast algorithm for finding dominators in a flowgraph. ACM Trans. Program. Lang. Syst. 1(1), 121–141 (1979)

[121] Leung, C.C., Chau, H.F.: Weighted assortative and disassortative networks model. Physica A 378(2), 591–602 (2007)

[122] Li, Q., Yang, J.: The grey complex networks and design of some topological indexes of grey indirect networks. International Journal of Nonlinear Science 5(1), 33–38 (2008)

[123] Liu, S., Cao, N., Lv, H.: Interactive visual analysis of the NSF funding information. In: IEEE Pacific Visualisation Symposium 2008. pp. 183–190 (2008)

[124] Liu, S., Lin, Y.: Grey Information: Theory and Practical Applications. Advanced Information and Knowledge Processing, Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)

[125] Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M.: Applying social network analysis to the information in CVS repositories. In: MSR 2004. pp. 101–105. IET Digital Library (2004)

[126] Lowry, E.S., Medlock, C.W.: Object code optimization. Communications of the ACM 12(1), 13–22 (1969)

[127] Lü, H., Fogarty, J.: Cascaded treemaps: examining the visibility and stability of structure in treemaps. In: ACM Proceedings of graphics interface 2008. pp. 259–266 (2008)

[128] Lütkepohl, H.: New Introduction to Multiple Time Series Analysis. Springer-Verlag (2006)

[129] Maritan, A., Banavar, J.R.: Chaos, noise, and synchronization. Physical Review Letters 72(10), 1451–1454 (1994)

[130] Marple, S.L.: Digital spectral analysis: with applications. Prentice-Hall (1987)

[131] Maslov, S., Sneppen, K.: Specificity and stability in topology of protein networks. Science 296(5569), 910–913 (2002)

[132] Massimini, M., Ferrarelli, F., Huber, R., Esser, S.K., Singh, H., Tononi, G.: Breakdown of cortical effective connectivity during sleep. Science 309(5744), 2228–2232 (2005)

[133] McSharry, P.E., Smith, L.A., Tarassenko, L.: Comparison of predictability of epileptic seizures by a linear and a nonlinear method. IEEE Transactions on Biomedical Engineering 50(5), 628–633 (2003)

[134] Meunier, D., Achard, S., Morcom, A., Bullmore, E.T.: Age-related changes in modular organization of human brain functional networks. NeuroImage 44(3), 715–723 (2009)

[135] Meunier, D., Lambiotte, R., Fornito, A., Ersche, K.D., Bullmore, E.T.: Hierarchical modularity in human brain functional networks. Frontiers in Neuroinformatics 3(37) (2009)

[136] Mima, T., Matsuoka, T., Hallett, M.: Functional coupling of human right and left cortical motor areas demonstrated with partial coherence analysis. Neuroscience Letters 287(2), 93–96 (2000)

[137] Mitschke, F.: Acausal filters for chaotic signals. Physical Review A 41(2), 1169–1171 (1990)

[138] Mormann, F., Lehnertz, K., David, P., Elger, C.E.: Mean phase coherence as a measure for phase synchronization and its application to the EEG of epilepsy patients. Physica D 144(3-4), 358–369 (2000)

[139] Morrison, J.L., Breitling, R., Higham, D.J., Gilbert, D.R.: Generank: Using search engine technology for the analysis of microarray experiments. BMC Bioinformatics 6(233), 1–14 (2005)

[140] Mountcastle, V.B.: Perceptual Neuroscience: The Cerebral Cortex. Harvard University Press (1998)

[141] Mowbray, M., Lain, A.: Dominator-tree analysis for distributed authorization. In: PLAS '08. pp. 101–112. ACM (2008)

[142] Muchnick, S.S.: Advanced compiler design and implementation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997)

[143] Munn, K., Smith, B. (eds.): Applied Ontology: An Introduction, Metaphysical Research, vol. 9. ontos verlag, Frankfurt/Lancaster (2008)

[144] Newman, M.E.J.: Assortative mixing in networks. Physical Review Letters 89(20), 208701 (2002)

[145] Newman, M.E.J.: Mixing patterns in networks. Physical Review E 67(2), 026126 (2003)

[146] Newman, M.E.J., Park, J.: Why social networks are different from other types of networks. Physical Review E 68(3), 036122 (2003)

[147] Newman, M.E.J., Watts, D.J., Strogatz, S.H.: Random graph models of social networks. Proceedings of the National Academy of Sciences USA 99, 2566–2572 (2002)

[148] Nolte, G., Bai, O., Wheaton, L., Mari, Z., Vorbach, S., Hallett, M.: Identifying true brain interaction from EEG data using the imaginary part of coherency. Clinical neurophysiology 115(10), 2292–2307 (2004)

[149] Oikonomou, T.: InBrAiN: An Interactive Tool for Brain Analysis and Visualization. Master's thesis, Computer Science Department, University of Crete (2007), http://elocus.lib.uoc.gr/dlib/5/d/7/attached-metadata-dlib-2007oikonomou/2007oikonomou.pdf

[150] Onnela, J.P., Saramäki, J., Kertész, J., Kaski, K.: Intensity and coherence of motifs in weighted complex networks. Physical Review E 71(6), 065103 (2005)

[151] Oostenveld, R., Praamstra, P.: The five percent electrode system for high-resolution EEG and ERP measurements. Clinical Neurophysiology 112(4), 713–719 (2001)

[152] Oshio, K., Morita, S., Osana, Y., Oka, K.: C. elegans synaptic connectivity data. Tech. Rep. 1, CCeP, Keio Future, Keio University (1998)

[153] Pachou, E., Vourkas, M., Simos, P., Smit, D., Stam, C.J., Tsirka, V., Micheloyannis, S.: Working memory in schizophrenia: An EEG study using power spectrum and coherence analysis to estimate cortical activation and network behavior. Human Brain Mapping 21(2), 128–137 (2008)

[154] Packard, N.H., Crutchfield, J.P., Farmer, J.D., Shaw, R.S.: Geometry from a time series. Physical Review Letters 45(9), 712–716 (1980)

[155] Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)

[156] hyun Park, C., Kim, S.Y., Kim, Y.H., Kim, K.: Comparison of the small-world topology between anatomical and functional connectivity in the human brain. Physica A 387(23), 5958–5962 (2008)

[157] Pecora, L.M., Carroll, T.L.: Synchronization in chaotic systems. Physical Review Letters 64(8), 821–824 (1990)

[158] Pikovsky, A., Rosenblum, M., Kurths, J.: Synchronization: A Universal Concept in Nonlinear Sciences. Cambridge University Press (2001)

[159] Prosser, R.T.: Applications of boolean matrices to the analysis of flow diagrams. In: Proc. Eastern Joint Computer Conf. Dec. 1959. pp. 133–138 (1959)

[160] Pyragas, K.: Weak and strong synchronization of chaos. Physical Review E 54(5), R4508–R4511 (1996)

[161] Quiroga, R.Q., Kraskov, A., Kreuz, T., Grassberger, P.: Performance of different synchronization measures in real data: A case study on electroencephalographic signals. Physical Review E 65(4), 041903 (2002)

[162] Ravasz, E., Somera, A.L., Mongru, D.A., Oltvai, Z.N., Barabási, A.L.: Hierarchical organization of modularity in metabolic networks. Science 297(5586), 1551–1555 (2002)

[163] Robin, W.: Four Colors Suffice. Penguin Books Ltd, London (2002)

[164] Rodríguez-Velázquez, J.A., Estrada, E., Gutiérrez, A.: Functional centrality in graphs. Linear and Multilinear Algebra 55(3), 293–302 (2007)

[165] Romanik, K.: Directed VR-representable graphs have unbounded dimension. In: GD '94. pp. 177–181. Springer-Verlag, London, UK (1995)

[166] Rosenblum, M., Pikovsky, A., Kurths, J., Schäfer, C., , Tass, P.A.: Phase synchronization: from theory to data analysis. In: Neuro-Informatics and Neural Modelling, Handbook of Biological Physics, vol. 4. North Holland (2001)

[167] Rosenblum, M.G., Pikovsky, A.S., Kurths, J.: From phase to lag synchronization in coupled chaotic oscillators. Physical Review Letters 78(22), 4193–4196 (1997)

[168] Rubinov, M., Knock, S.A., Stam, C.J., Micheloyannis, S., Harris, A.W.F., Williams, L.M., Breakspear, M.: Small-world properties of nonlinear brain activity in schizophrenia. Human Brain Mapping 30(2), 403–416 (2009)

[169] Rulkov, N.F., Sushchik, M.M., Tsimring, L.S., Abarbanel, H.D.: Generalized synchronization of chaos in directionally coupled chaotic systems. Physical Review E 51(2), 980–994 (1995)

[170] Sakkalis, V., Giurcăneanu, C.D., Xanthopoulos, P., Zervakis, M., Tsiaras, V., Yang, Y., Karakonstantaki, E., Micheloyannis, S.: Assessment of linear and nonlinear synchronization measures for analyzing EEG in a mild epileptic paradigm. IEEE Transactions on Information Technology in Biomedicine 13(4), 433–441 (2009)

[171] Sakkalis, V., Oikonomou, T., Tsiaras, V., Zervakis, M., Tollis, I.: Graph-theoretic indices of evaluating brain network synchronization: Application in an alcoholism paradigm (2009), in preparation

[172] Sakkalis, V., Tsiaras, V., Michalopoulos, K., Zervakis, M.: Assessment of neural dynamic coupling and causal interactions between independent EEG components from cognitive tasks using linear and nonlinear methods. In: IEEE EMBC 2008. pp. 3767–3770 (2008)

[173] Sakkalis, V., Tsiaras, V., Zervakis, M., Tollis, I.G.: Optimal brain network synchrony visualization: Application in an alcoholism paradigm. In: IEEE EMBC 2007. pp. 4285–4288 (2007)

[174] Sauer, T., Yorke, J.A., Casdagli, M.: Embedology. Journal of Statistical Physics 65(3-4), 579–616 (1991)

[175] Schank, T., Wagner, D.: Approximating clustering coefficient and transitivity. Journal of Graph Algorithms and Applications 9(2), 265–275 (2005)

[176] Schelter, B., Winterhalder, M., Eichler, M., Peifer, M., Hellwig, B., Guschlbauer, B., Lücking, C.H., Dahlhaus, R., Timmer, J.: Testing for directed influences among neural signals using partial directed coherence. Journal of Neuroscience Methods 152(1-2), 210–219 (2006)

[177] Schelter, B., Timmer, J., Eichler, M.: Assessing the strength of directed influences among neural signals using renormalized partial directed coherence. Journal of Neuroscience Methods 179(1), 121–130 (2009)

[178] Schlögl, A.: A comparison of multivariate autoregressive estimators. Signal Processing 86(9), 2426–2429 (2006)

[179] Schnitzler, A., Gross, J.: Normal and pathological oscillatory communication in the brain. Nature Reviews Neuroscience 6(4), 285–296 (2005)

[180] Schwikowski, B., Uetz, P., Fields, S.: A network of protein-protein interactions in yeast. Nature Biotechnology 18(12), 1257–1261 (2000)

[181] Sharir, M.: Structural analysis: A new approach to flow analysis in optimizing compilers. Computer Languages 5(3-4), 141–153 (1980)

[182] Shen, Z.C., Chu, C.C.N.: Bounds on the number of slicing, mosaic, and general floorplans. IEEE Trans. on CAD of Integrated Circuits and Systems 22(10), 1354–1361 (2003)

[183] Shneiderman, B.: Tree visualization with tree-maps: 2-d space-filling approach. ACM Transactions on Graphics 11(1), 92–99 (1992)

[184] Sinatra, R., De Vico Fallani, F., Astolfi, L., Babiloni, F., Cincotti, F., Latora, V., Mattia, D.: Cluster structure of functional networks estimated from high-resolution EEG data. International Journal of Bifurcation and Chaos 19(2), 665–676 (2009)

[185] Snodgrass, J.G., Vanderwart, M.: A standardized set of 260 pictures: Norms for name agreement, image agreement, familiarity, and visual complexity. Journal of Experimental Psychology: Learning, Memory, and Cognition 6(2), 174–215 (1980)

[186] Sporns, O., Honey, C.J., Kötter, R.: Identification and classification of hubs in brain networks. PLoS ONE 2(10) (2007)

[187] Sporns, O., Tononi, G., Kötter, R.: The human connectome: A structural description of the human brain. PLoS Computational Biology 1(4), e42 (2005)

[188] Stam, C.J., van Dijk, B.W.: Synchronization likelihood: an unbiased measure of generalized synchronization in multivariate data sets. Physica D 163(3), 236–251 (2002)

[189] Stam, C.J., Jones, B., Nolte, G., Breakspear, M., Scheltens, P.: Small-world networks and functional connectivity in alzheimer's disease. Cerebral Cortex 17(1), 92–99 (2007)

[190] Stam, C.J., Pijn, J.P., Suffczynski, P., da Silva, F.H.L.: Dynamics of the human alpha rhythm: evidence for non-linearity? Clinical Neurophysiology 110(10), 1801–1813 (1999)

[191] Stam, C.J., Reijneveld, J.C.: Graph theoretical analysis of complex networks in the brain. Nonlinear Biomedical Physics 1(3) (2007)

[192] Strogatz, S.H., Stewart, I.: Coupled oscillators and biological synchronization. Scientific American 269(6), 102–109 (1993)

[193] Sugeno, M.: Fuzzy measures and fuzzy integrals: a survey. In: Gupta, M., Saridis, G., Gains, B. (eds.) Fuzzy automata and decision processes, pp. 89–102. North Holland, Amsterdam (1977)

[194] Symeonidis, A., Tollis, I.G., Reczko, M.: Visualization of functional aspects of microRNA regulatory networks using the gene ontology. In: ISBMDA. LNCS, vol. 4345, pp. 13–24. Springer (2006)

[195] Takahashi, D.Y., Baccalá, L.A., Sameshima, K.: Connectivity inference between neural structures via partial directed coherence. Journal of Applied Statistics 34(10), 1259–1273 (2007)

[196] Takahashi, D.Y., Baccalá, L.A., Sameshima, K.: Partial directed coherence asymptotics for VAR processes of infinite order. International Journal of Bioelectromagnetism 10(1), 31–36 (2008)

[197] Takens, F.: Detecting strange attractors in turbulence. In: Dynamical Systems and Turbulence. Lecture Notes in Mathematics, vol. 898, pp. 366–381 (1981)

[198] Tamassia, R., Preparata, F.P.: Dynamic maintenance of planar digraphs, with applications. Algorithmica 5(1), 509–527 (1990)

[199] Tamassia, R., Tollis, I.G.: A unified approach to visibility representations of planar graphs. Discrete and Computational Geometry 1(1), 321–341 (1986)

[200] Tanaka, R.: Scale-rich metabolic networks. Physical Review Letters 94(16), 168101 (2005)

[201] Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)

[202] Theiler, J.: Spurious dimension from correlation algorithms applied to limited time-series data. Physical Review A 34(3), 2427–2432 (1986)

[203] Thomas W. Valente, R.K.F.: Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. Social Networks 20(1), 89–105 (1998)

[204] Tsiaras, V., Andreou, D., Tollis, I.G.: Brainnetvis: Analysis and visualization of brain functional networks. In: IEEE EMBC 2009. pp. 2911–2914 (2009)

[205] Tsiaras, V., Tollis, I.G.: Brain network analyzer. In: Tollis, I.G., Patrignani, M. (eds.) Graph Drawing: GD 2008. LNCS, vol. 5417, pp. 451–452. Springer-Verlag (2009)

[206] Tsiaras, V., Tollis, I.G.: Dagmap View. In: Tollis, I.G., Patrignani, M. (eds.) Graph Drawing: GD 2008. LNCS, vol. 5417, pp. 449–450. Springer-Verlag (2009)

[207] Tsiaras, V., Triantafilou, S., Tollis, I.G.: Dagmaps: Space filling visualization of directed acyclic graphs. Journal of Graph Algorithms and Applications 13(3), 319–347 (2009)

[208] Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series parallel digraphs. SIAM Journal on Computing 11(2), 298–313 (1982)

[209] Vincent, J.L., Patel, G.H., Fox, M.D., Snyder, A.Z., Baker, J.T., Essen, D.C.V., Zempel, J.M., Snyder, L.H., Corbetta, M., Raichle, M.E.: Intrinsic functional architecture in the anaesthetized monkey brain. Nature 447(7140), 83–86 (2007)

[210] Wang, Y., Berg, P., Scherg, M.: Common spatial subspace decomposition applied to analysis of brain responses under multiple task conditions: a simulation study. Clinical Neurophysiology 110(4), 604–614 (1999)

[211] Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393(6684), 440–442 (1998)

[212] White, J.G., Southgate, E., Thomson, J.N., Brenner, S.: The structure of the nervous system of the nematode caenorhabditis elegans. Philosophical Transactions of the Royal Society of London - Series B 314(1165), 1–340 (1986)

[213] van Wijk, J.J., van de Wetering, H.: Cushion treemaps: Visualization of hierarchical information. In: INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization. pp. 73–78 (1999)

[214] Winterer, G., Enoch, M.A., White, K.V., Saylan, M., Coppola, R., Goldman, D.: EEG phenotype in alcoholism: increased coherence in the depressive subtype. Acta Psychiatrica Scandinavica 108(1), 51–60 (2003)

[215] Wismath, S.K.: Characterizing bar line-of-sight graphs. In: SCG '85: Proceedings of the first annual symposium on Computational geometry. pp. 147–152. ACM (1985)

[216] Xulvi-Brunet, R., Sokolov, I.M.: Reshuffling scale-free networks: From random to assortative. Physical Review E 70(6), 066102 (2004)

[217] Yager, R.R.: On the measure of fuzziness and negation part I: Membership in the unit interval. International Journal of General Systems 5(4), 221–229 (1979)

[218] Yip, A.M., Horvath, S.: Gene network interconnectedness and the generalized topological overlap measure. BMC Bioinformatics 8(22) (2007)

[219] Yu, H., Kim, P.M., Sprecher, E., Trifonov, V., Gerstein, M.: The importance of bottlenecks in protein networks: Correlation with gene essentiality and expression dynamics. PLoS Computational Biology 3(4), e59 (2007)

[220] Zhang, B., Horvath, S.: A general framework for weighted gene coexpression network analysis. Statistical Applications in Genetics and Molecular Biology 4(1) (2005)

[221] Zhang, X.L., Begleiter, H., Porjesz, B., Wang, W., Litke, A.: Event related potentials during object recognition tasks. Brain Research Bulletin 38(6), 531–538 (1995)

[222] Zhao, X.M., Wang, R.S., Chen, L., Aihara, K.: Uncovering signal transduction networks from high-throughput data by integer linear programming. Nucleic Acids Research 36(9), e48 (2008)

Chapter 10

Appendices

10.1 Appendix I: Eigenvalue Computation

Power iteration is one of many eigenvalue algorithms that may be used to find this dominant eigenvector (see Algorithm 16). This algorithm, which is a Monte Carlo algorithm, fails to converge when there are more than one principal eigenvalues. For example a star graph has one positive and one negative eigenvalues that have the same magnitude, and all other eigenvalues are zero. The convergence to the positive eigenvalue is restored if we add a diagonal matrix with very small positive and random entries to adjacency matrix A. However the convergence rate is slow since it depends on the ratio $\frac{\lambda_2}{\lambda_1}$. To improve the convergence ratio we propose the Monte Carlo algorithm proposed by Booth [40] in the context of physics and is explained in Gubernatis and Booth [85]. His method returns both principal eigenpairs (λ_1, v_1) and (λ_2, v_2) and the convergence rate for the first eigenpair depends on the ratio $\frac{\lambda_3}{\lambda_1}$.

Algorithm 16 Power iteration for computing the largest eigenvalue

Input: Matrix $W \in \mathbb{R}^{n \times n}$ and vector $\mathbf{q}^{(0)}$ with $\| \mathbf{q}^{(0)} \| = 1$
Output: Largest eigenvalue $\lambda^{(k)}$ in absolute value
and corresponding eigenvector $\mathbf{q}^{(k)}$

$k \leftarrow 1$;
repeat
  $\mathbf{z}^{(k)} \leftarrow W\mathbf{q}^{(k-1)}$
  $\mathbf{q}^{(k)} \leftarrow \mathbf{z}^{(k)} / \| \mathbf{z}^{(k)} \|_2$
  $k \leftarrow k + 1$
until $\mathbf{q}^{(k)}$ is acceptable approximation
$\lambda^{(k)} \leftarrow (\mathbf{q}^{(k)})^T W \mathbf{q}^{(k)}$

10.2 Appendix II: Squarified Layout Algorithm

Algorithm 17 Squarified Layout

Input: Rectangle r, numeric parameter $pSize$, array of nodes $children$
Output: divide rectangle r into sub-rectangles assigned to nodes in $children$

sortDescending($children$); // sort children list according to $size$ in descending order
squarifiedLayout(r, $pSize$, $children$);

function squarifiedLayout(r, $pSize$, $children$)
$x \leftarrow r.x$; $y \leftarrow r.y$; $w \leftarrow r.w$; $h \leftarrow r.h$;
$startBlock \leftarrow$ **true**; $i \leftarrow 0$; $numChildren \leftarrow children.length$;
while ($i < numChildren$) **do**
if ($startBlock$) **then** //Book keeping the new block
$isAxisVertical \leftarrow (w < h)$;
$startBlock \leftarrow$**false**; $endBlock \leftarrow$ **false**; $blockFrom \leftarrow i$;
$bestAspectRatio \leftarrow +\infty$; $blockSize \leftarrow 0$;
// Inside block
$newSize \leftarrow children[i].size$;
$blockSize \leftarrow blockSize + newSize$;
if ($isAxisVertical$) **then**
$bw \leftarrow (newSize/blockSize) * w$; $bh \leftarrow (blockSize/pSize) * h$;
else
$bw \leftarrow (blockSize/pSize) * w$; $bh \leftarrow (newSize/blockSize) * h$;
$newAspectRatio \leftarrow \max(bw/bh,\ bh/bw)$;
if ($newAspectRatio < bestAspectRatio$) **then**
$bestAspectRatio \leftarrow newAspectRatio$;
if ($i = (numChildren - 1)$) **then**
$blockTo \leftarrow numChildren - 1$; $endBlock \leftarrow$ **true**;
$i \leftarrow i + 1$;
else
$blockTo \leftarrow i - 1$; $blockSize \leftarrow blockSize - newSize$; $endBlock \leftarrow$ **true**;
// Assign rectangles to vertices of the block
if ($endBlock$) **then**
$childWidth \leftarrow w$; $childHeight \leftarrow h$; $childX \leftarrow x$; $childY \leftarrow y$;
if ($isAxisVertical$) **then**
$childHeight \leftarrow (blockSize/pSize) * h$;
else
$childWidth \leftarrow (blockSize/pSize) * w$;
for $j \leftarrow blockFrom$ **to** $blockTo$ **do**
$children[j].r \leftarrow$ new $Rectangle(childX, childY, 0, 0)$;
$proportion = children[j].size/blockSize$;
if ($isAxisVertical$) **then**
$childWidth \leftarrow proportion * w$; $childX \leftarrow childX + childWidth$;
else
$childHeight \leftarrow proportion * h$; $childY \leftarrow childY + childHeight$;
$children[j].r.w \leftarrow childWidth$; $children[j].r.h \leftarrow childHeight$;
if ($isAxisVertical$) **then**
$y \leftarrow y + childHeight$; $h \leftarrow h - childHeight$;
else
$x \leftarrow x + childWidth$; $w \leftarrow w - childWidth$;
$pSize \leftarrow pSize - blockSize$; $startBlock \leftarrow$**true**;