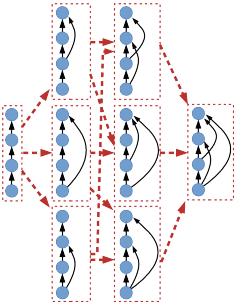
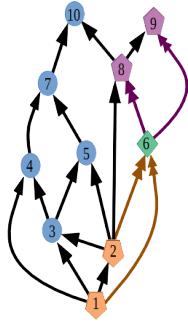


Confidential.

Not to be redistributed without permission of the author.

17th June 2025

(dagology.tex LaTeX-ed on June 17, 2025)



DAGOLOGY

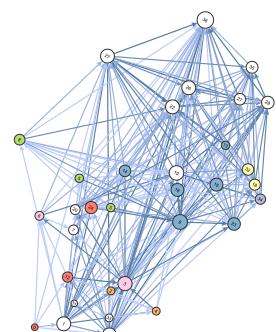
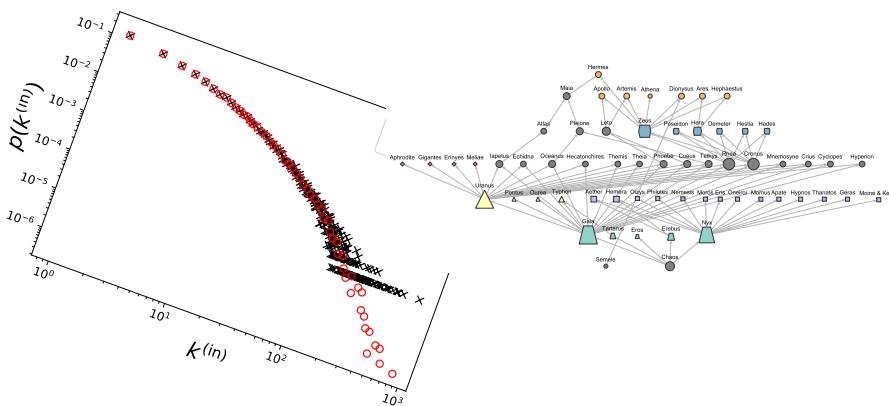
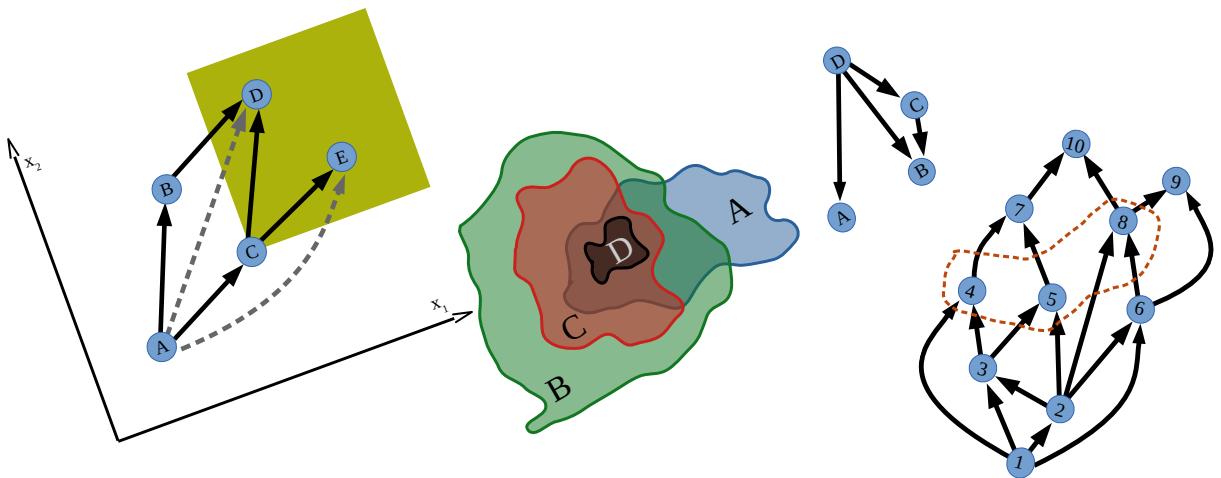
An Overview of Directed Acyclic Graphs

Tim S. Evans

Centre for Complexity Science & Physics Department,
Imperial College London.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam vehicula leo id nisi volutpat convallis. Aliquam quis nunc orci. Accedite et ecce arbor in. Puella si potes invenire. Accedite et ecce tenebræ. Nam post te. Audio vocem invocabat nomen meum. Alta voce in tenebris. Arboribus primum occurrit vocem audio. Sed quia repente serum tincidunt. Silva perditus sum solus. Puella non est, semper est idem. Non currit contra me, iterum atque iterum atque iterum.



Contents

Contents	2
List of Figures	4
List of Tables	6
1 Network Basics	11
1.1 What is a network?	11
1.2 Basic network measurements	15
1.3 Centrality	21
1.4 Processes on networks	23
1.5 Classic network models	27
2 DAG Properties	33
2.1 What is a DAG?	33
2.2 Local Measures	35
2.3 Useful DAG Subgraphs	37
2.4 Paths	39
2.5 Height and Depth	41
2.6 Total Order	42
2.7 Transitive Reduction and Transitive Completion.	46
2.8 The Adjacency Matrix of a DAG	52
2.9 Posets	55
2.10 Chains	59
2.11 Trees	63
2.12 DAGlets - small DAGs	63
2.13 Temporal Networks	68
2.14 Other DAG features	69
3 Analysing DAGs	71
3.1 DAG Centrality	72
3.2 Edge and Path analysis for DAGs	100
3.3 The distance between nodes	105
3.4 Layering	109
4 DAG Models	111
4.1 Random Order DAG	111
4.2 Set Inclusion	112
4.3 The Poisson Point Process for nodes	114
4.4 Cube-Space DAG	116
4.5 The Random Geometric Graph model	118

4.6	Minkowski Space-Time DAG	120
4.7	Pyramid and Disk Space DAGs	124
4.8	The Price model	129
4.9	Other Citation Models	142
5	Wild DAGs	143
5.1	Citation Networks	144
5.2	Scheduling	145
5.3	Food Webs	150
5.4	Ratings and Rankings	156
5.5	Bayesian Networks and Causal Inference	157
5.6	Spreadsheets	162
5.7	Genealogy Trees	164
5.8	Hierarchies including voltage graphs	164
5.9	Data Compression	165
5.10	Geometric embedding of data	165
5.11	Sampling Data for a DAG	165
5.12	Null Models and Data Randomisation	166
6	DAG Numerics	167
6.1	The Numerical Representation of DAGs	168
6.2	Numerical Algorithms for DAGs	170
6.3	Randomising DAGs	173
6.4	Numerical Implementation of Price model	177
6.5	Visualisation	178
6.6	Removing Cycles	179
6.7	Hierarchies, Trees, Hyperbolic Space and DAGs	185
Bibliography		187
A	Useful Mathematics	201
A.1	Linear Algebra	201
A.2	Generating Functions	202
A.3	The Gamma and Beta functions	204
A.4	Sets	206
B	Additional Material	209
B.1	Disk Spaces and Formal Balls	209
C	Unused Material	211
C.1	Additional Network Material	211
C.2	Cube-Space DAG	212
C.3	Box Space	217
C.4	Extended Family of Metrics	222
C.5	Disk Spaces and Box Spaces	224
C.6	XTent model	225
C.7	Boxicity	226
C.8	Application of Box Spaces	230

List of Figures

1.1	An example of a network, nodes, edges, and components.	11
1.2	A subgraph illustration.	13
1.3	Examples of different types of network.	14
1.4	The convention used for adjacency matrices.	14
1.5	Some example adjacency matrices.	15
1.6	Examples of fat-tailed degree distributions from real datasets.	16
1.7	Examples of a walk, a cycle and a path.	17
1.8	Examples of directed path.	18
1.9	An example of a tree.	18
1.10	Illustration of components of a network.	21
1.11	Strongly and weakly connected components of a directed network.	22
1.12	Illustration of one iteration of a broadcast process.	24
1.13	Illustration of one iteration of a diffusion process.	25
1.14	Sketch of different degree distributions.	30
1.15	Illustration of no node correlation assumption.	31
1.16	Numerical implementation of configuration model from a list of degrees	31
1.17	Illustration of how to rewire a pair of edges which preserves the degree of nodes.	32
2.1	Two directed networks, one a DAG, one not.	34
2.2	Some simple illustrations of real world DAGs.	34
2.3	An illustration of the past neighbourhood and the future neighbourhood of a node.	36
2.4	An illustration of the past and future DAGs.	38
2.5	An illustration of an interval.	38
2.6	An illustration of the longest paths in a DAG.	40
2.7	An illustration of height, depth and criticality of nodes in a DAG.	43
2.8	An illustration of a topological sort algorithm.	45
2.9	An illustration of height of nodes in a DAG.	46
2.10	Hierarchies.	47
2.11	A transitively complete DAG.	47
2.12	A transitively reduced DAG.	48
2.13	An illustration of a DAG of DAGs.	49
2.14	An illustration of a family of related DAGs.	50
2.15	An illustration of a DAG of DAGs.	51
2.16	Another illustration of the network length of an edge in a DAG.	51
2.17	An illustration a network with a path made up only of essential edges that is not a longest path.	52
2.18	The adjacency matrix A for and example DAG.	53
2.19	The Möbius inversion M for an example DAG.	55
2.20	Two visualisations of a poset.	57
2.21	One visualisation of a poset.	59
2.22	An illustration of the properties of a chain.	60

2.23 An illustration of a cover of chains.	61
2.24 An illustration of an antichain.	62
2.25 Example of an antichain.	63
2.26 The two unlabelled and three labelled DAGs with two nodes.	64
2.27 Two labellings with the same labelled DAG.	65
2.28 One unlabelled DAG from three different triangular matrices.	66
2.29 The four unlabelled DAGS with three nodes and one weakly connected component.	67
2.30 All ten transitively reduced unlabelled DAGS with four nodes and one component..	67
2.31 Example of a line graph.	69
2.32 Example of a line graph of a temporal network.	69
3.1 The average neighbour value node measures.	90
3.2 An illustration of height, depth and criticality of nodes in a DAG.	101
3.3 Historiogram of DNA.	104
3.4 Example of the shortest path.	107
3.5 An illustration of the network length of an edge in a DAG.	109
4.1 An example of the construction of a Random Order DAG.	112
4.2 An example of a set inclusion DAG.	113
4.3 An example of a set inclusion DAG given in terms of circles.	113
4.4 Disk space.	114
4.5 An example of a set inclusion DAG created in a geometric context.	114
4.6 An example of a cube-space DAG.	116
4.7 A random geometric graph.	119
4.8 An ordered random geometric graph.	120
4.9 A directed random geometric graph.	121
4.10 The light cones for a Minkowski space-time DAG.	123
4.11 The backwards connection kernels in a square pyramid model DAG.	125
4.12 The XTent model.	128
4.13 The relationship between cube and disk space in two dimensions.	130
4.14 The fat-tailed degree distribution in a citation network.	131
4.15 How the network grows in the Price model.	132
4.16 The degree distribution from a numerical simulations of the Price model.	137
5.1 The definition of a citation network.	144
5.2 Citation distribution before and after transitive reduction.	145
5.3 Programming language historiograph.	147
5.4 DAG of python package dependency on one PC.	149
5.5 An example of a cube space criteria used to combine ranks.	157
5.6 A Causal Inference DAG.	160
6.1 The three edge swapping possibilities in a DAG.	174
6.2 Example of DAG rewiring.	175
6.3 DAG edge swapping while preserving height.	175
6.4 How the attachment list works in the Price model.	178
6.5 Using height and depth as coordinates for a DAG.	179
6.6 The DAG produced from the Zachary karate club network using degree as an order.	180
C.1 A cube-space DAG.	214
C.2 The forward connection kernels in a cube-space DAG.	214
C.3 An illustration of a measure. [From Wikipedia].	218
C.4 An illustration of theorem 1 of Bollobás and Brightwell (1991).	220

C.5	An illustration of theorem 2 of (Bollobás and Brightwell, 1991).	220
C.6	The relationship between cube and disk space in two dimensions.	225
C.7	An example of transitive reduction and transitive completion.	228
C.8	Example of how boxicity constructions are mapped onto DAGs.	228
C.9	How images are ordered as a DAG using textual terms.	230

List of Tables

1.1	Different objects and relationships captured by nodes and edges.	12
1.2	Different network terminology found in different fields.	12
2.1	Number of DAGlets	65
2.2	Numbers of different types of DAGlets.	68
3.1	Node centrality values.	74
5.1	Information on the nodes on a longest path in the programming language DAG. Only nodes of centrality zero, i.e. all nodes that lie on a longest path, are listed.	148

Construction Notes

Outline

- Introduction
- Basic Ordinary Network Properties
 - Definition of Network, Simple Network and Directed Network
 - Paths, Shortest Path, Diameter, cycles
 - Connectivity & Components, Strong and Weak
- DAG Properties
 - Definition: Simple by example, then formal.
 - Examples and Applications (brief)
 - Sources and Sinks
 - Chains and paths, height and depth
 - Longest Path
 - Chains and Height. Transitive Reduction and Transitive Completion. Visualisation
 - Hasse Diagram. Essential edges (Links) and non-essential edges
 - Antichains, width, Dilworth Theorem.
 - Relation of DAGs to Partially Ordered Sets
- DAG Models
 - Set Inclusion
 - Random Orders
 - Price model (BA model)
 - Food Web models (interval graphs?)
 - Poisson Point Process models/Box spaces:
 - * Cube Space, Random orders
 - * Minkowski (flat) space-time model, geodesics, comparison with shortest path and Euclidean space
 - * de Sitter and curved space models
 - Citation Network Models
 - Configuration Model for DAGs
- DAGs in the wild: Data Sets
 - Citation Networks: Academic papers, patents, court judgements
 - Food Webs
 - Computer Packages
 - Ratings and Rankings
 - Bayesian Networks
 - Spreadsheets

- Genealogy Trees
- Approximate DAGs
- DAG Numerics
 - Removing cycles from directed graphs
 - Longest Path algorithm
 - Antichain Algorithm
 - Visualisation and Layout
 - Agony

Style and Notation

Overall style is for a book aimed at my first year PhD students. Additional maths can be in an appendix or reserved for a review. Mathematical notation is for precision and brevity rather than for rigorous proof. Any mathematical notation should be backed in (i) words, (ii) figures and (iii) examples.

Capitalisation?

Title case is also called headline style or up style: you see it used in headings and titles of books, movies, TV shows, articles, and other works. Sentence case, also called sentence style or down style, is used for second-level headings and lower. News headlines have traditionally been capitalized using title case, although these days, sentence case is often used, especially online.

- Book title in **title case**.
- Chapter and section headings in **sentence case**.
- Model, theorem names in **???**.
Is it “Cube Space model” or “cube-space model” or “Cube-space model”? As theorem is lower case, then model should be. Does the name of a model/theorem become a proper noun, a name, and so is capitalised under sentence case rules?

Preferred terms

- **node** rather than vertex. Preferred as vertex sounds more technical and mathematical.
- **edge** not link. (**T**)¹
- **network** preferred to graph (the latter sounds very technical) but latter is used when in a standard name.
- **successors** and **predecessors** rather than children and parents for nodes in the future and past vertex set. (**T**) Surely this is inconsistent with next definition? (**end T**)
- **descendents** and **ancestors** for nearest nodes in the future and past vertex set. (**T**)²

¹(**T**) We could switch to link which matches node better, node is less technical too?

²(**T**) Is this genealogy tree language right if we don't use children and parents for nearest neighbours?

- **past** and **future** as subgraphs including source/sink. These sub-DAGs have vertex sets that include the root node and we define some notation that exclude the root node, i.e. the descendent and ancestor node sets. Can also use \prec and \preceq to define these.
- Use of + (–) superscript without brackets to indicate something to do with a vertex v that is related to the future (past) of v e.g. X_v^\pm .
- The exception is degree $k_v^{(\text{in})} \equiv k_v^-$ which is the number of nearest neighbours in the past while $k_v^{(\text{out})} \equiv k_v^+$ is the number of neighbours in the future.
- TO DECIDE. We use the labels sp and lp for longest and shortest path vs min and max on path lengths, distances, secondary quantities defined in terms of lengths and distances. Idea use sp/lp for actual primary cases where these are used. Use max/min for distance and secondary quantities where we may use max/min of distance/lengths other than network lengths.

Preferred style

- No definition environment
- Minimise number of theorem environments
- Larger examples as separate environment?

Symbols

Symbols	Usage
u, v	Vertices
$\{\dots\}$	set
$(v_0, v_1, \dots, v_L) \equiv (v_n)_{n=0}^L$	sequence

Mathematical Conventions

- Indices i, j for non-negative integers from a sequence $[0, (N - 1)]$ or $[1, N]$ or equivalent.
- Labels u, v for nodes from a set of nodes \mathcal{V} (V here is for vertex). (T)³
- Edges (u, v) run from node u to v to match sequences.
- Function arguments match matrices, so distance $d(v, u) \equiv d_{vu}$ is the distance from node u to node v (should this not be symmetric). Likewise the distance measured by the length of a shortest path is $d^{(\text{sp})}(v, u) \equiv d_{vu}^{(\text{sp})}$ is for the distance from u to v .
- Adjacency matrix A_{ij} describes an edge from j to i (see Fig. 1.4). Direction convention applies to all edge-based matrices.
- Adjacency matrix is strictly upper triangular, so $A_{ij} = 0$ if $i \geq j$, and $A_{ij} \neq 0$ only if $i < j$.
- The length of a walk/path $\ell(\mathcal{W})$ is the number of edges in a walk/path.
Use LENGTH for edge count length when no ambiguity, NETWORK LENGTH when want to be more precise.
- The weight of a path $\omega(\mathcal{W})$ for sum of edge weights in a walk.

³(T)Sholdd we change? No \mathcal{N} is neighbourhood.

- The distance between two nodes $d(u, v)$.
- For lengths $\ell(\mathcal{P})$ and weights $\omega(\mathcal{P})$ of paths/walks longest/shortest path superscripts are often not needed as the path/walk is specified.
Use superscripts to indicate the type of path weight used if needed e.g. (net) for edge count, but unclear if we have superscript overload in some cases. For some weights, e.g. (spe), the max/min sp/lp is implied so not needed, the spe superscript is enough.
- For distances d , we use superscripts to indicate the type of path used: (sp) for shortest paths, (lp) for longest paths — not min and max. For other weights, the max/min is implied e.g. (spe) implies a weight and maximisation.

Commands

Command	Usage
<code>\eqref{<label>}</code>	Equation reference
<code>\figref{<label>}</code>	Figure reference
<code>\secref{<label>}</code>	Section reference
<code>\tsedef{<text>}</code>	Definition (no environment)
<code>\tseabbv{<text>}</code>	Abbreviation
<code>\tseemph{<text>}</code>	Emphasis (e.g. a term used earlier)
For the index. Forms with <code>only</code> do not put anything in main text, they only add this label	
Command	Usage
<code>\figindexonly{<label>}</code>	Figure, index only
<code>\tabindexonly{<label>}</code>	Table, index only
<code>\secindexonly{<label>}</code>	Section, index only
<code>\exindexonly{<label>}</code>	Example, index only
to the index. <code>\figindex{<label>}</code>	Figure, text and index
	Table, text and index
	Section, text and index
	Example text and index
	index only, entry in bold
	index only, entry in italics
	index only, entry in small caps

Chapter 1

Network Basics

(T)¹

(T) Change i, j to u, v for nodes. What about indices? (end T)

Aims and Objectives: To give define a network and its key components. To explain why it can be useful. To define various types of networks. To give examples of these. To give standard representations of networks. To discuss the role of visualisation.

1.1 What is a network?

A loose definition of a NETWORK is that is is a set of objects, called NODES (also known as NODES), which are connected in pairs by EDGES (also called LINKS). A simple example is shown in Fig. 1.1.

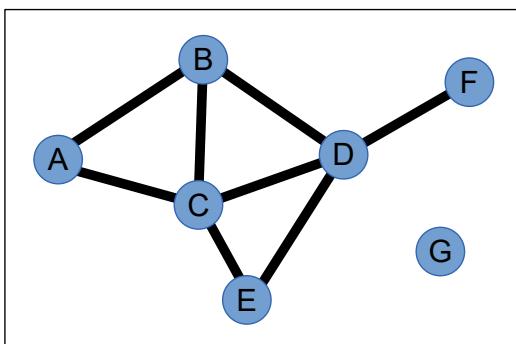


Figure 1.1: An example of a network of seven nodes and 8 edges. We may denote this as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where node set is $\mathcal{V} = \{A, B, C, D, E, F, G\}$ and the edge set, \mathcal{E} , is $\{(A, B), (A, C), (B, C), (B, D), (C, D), (C, E), (D, E), (D, F)\}$. Note that not all nodes need have an edge, and the whole network can have several disconnected parts called components (see section 1.2.4). In this case there are two components: nodes A to F and their edges form one component, while node G is a component by itself.

It is very natural to visualise a network displaying the nodes as points while lines between the points represent the edges, exactly as we have done in Fig. 1.1. However, nodes do not need to have any information about their location in some space. So we may place the nodes and edges on the page in any way we wish and we could display exactly the same network in many different way. The visualisation of networks is a big subject in its own right, as we will discuss in section 6.5.

A node can be any object you like but will typically be one person, one web page, one document, etc. The edges then record some type of relationship between pairs of the objects. Examples include a genetic relationship between two people, a hyperlink from one web page to another and a citation in one document of another document. The importance of relationships between pairs is widespread, as Table 1.1 suggests, making a network a useful representation in many different fields. Network science is all about these binary relationships between pairs of objects.

¹(T)Monday, December 31, 2018 08:42

However, the key point is that the study of networks is much more than a study of the statistics of pairs (Brandes et al., 2013). The real power of network analysis comes when we look at the wider picture, of how many different objects have indirect relationships, represented by different paths linking the objects. For instance in Fig. 1.1, nodes *A* and *F* have some sort of indirect relationship as there is no edge between them but there are several paths linking them. On the other hand, node *G* is completely disconnected and so has no direct or indirect relationship to any other node.

	network	node	edge
	social	person	friendship
	genetic	person	at least 50% shared DNA
	organisation	person	line of command
	world wide web	web pages	hyperlink
	internet	network router	communication link
	citation	paper	entry in bibliography
	protein network	proteins	participate in same reaction
	food web	species	predator-prey
	travel	location	trips made between locations
	streets	intersections	part of a street

Since a network is such a natural picture in so many different situations it used in many different areas and so, not surprisingly, the terminology used for networks varies considerably, as Table 1.2 shows. The utility of networks also means they have had a long history. In

“point”	“line”	“combination”	typical field
vertex	edge	graph	Mathematics
node	link	network	Engineering, Computer Science
site	bond	network	Physics
actor	tie	network	Social Science

Table 1.2: There is considerable variation in the terminology used for networks. This table shows some of the terms frequently encountered and the fields where they are often found.

mathematics, they stretch back to a famous discussion by Euler in 1736 of the different routes across the seven bridges of Konigsburg (modern Kaliningrad). Their usefulness to represent interactions between pairs of people means they have been used in social science since 1930’s (Freeman, 2004), for example see the SOCIOGRAMS in Jennings (1937), where the study of such networks is often called SOCIAL NETWORK ANALYSIS — SNA (Wasserman and Faust, 1994, Freeman, 2004, de Nooy et al., 2005, Borgatti et al., 2009).

(T) Do we use stubs? Yes later but do we really need them? It is also useful sometimes to refer to the two ‘ends’ of an edge and these are sometimes called STUBS. (end T)

A more formal notation is often not essential but is, at the very least, a useful shorthand. This notation is used in mathematics where a network is known as a GRAPH, which we will denote as denoted \mathcal{G} , and the subject is known as GRAPH THEORY. The formal mathematical definition of a network or network \mathcal{G} is² is that is is a pair of sets, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: a set of NODES (NODES) \mathcal{V} , and a set of EDGES (LINKS) \mathcal{E} . Each edge is a pair of nodes³ taken from \mathcal{V} , the set

²See appendix A.4 for a summary of set notation.

³We emphasise that in network science we are capturing pairwise relationships between nodes. If it is important to consider , say relationships between three or more nodes, these are represented by HYPEREDGES and gives us a HYPERGRAPH but we will not be using these structures here.

of possible nodes of the network. We will denote an edge between nodes u and v as (u, v) . In these notes we will also use $N = |\mathcal{V}|$ for the number of nodes, and $E = |\mathcal{E}|$ for the number of edges.

It is also useful to define a **SUBGRAPH**. A subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ of a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a *graph* made from a subsets of the node and edge sets of original network \mathcal{G} . However note that these subsets \mathcal{V}' and \mathcal{E}' *must* form valid network themselves where the critical issue is that the edges in the subgraph must be made from the nodes in the subgraph. That is if $(u, v) \in \mathcal{E}'$ then $u, v \in \mathcal{V}'$ is necessary. See Fig. 1.2 for an example.

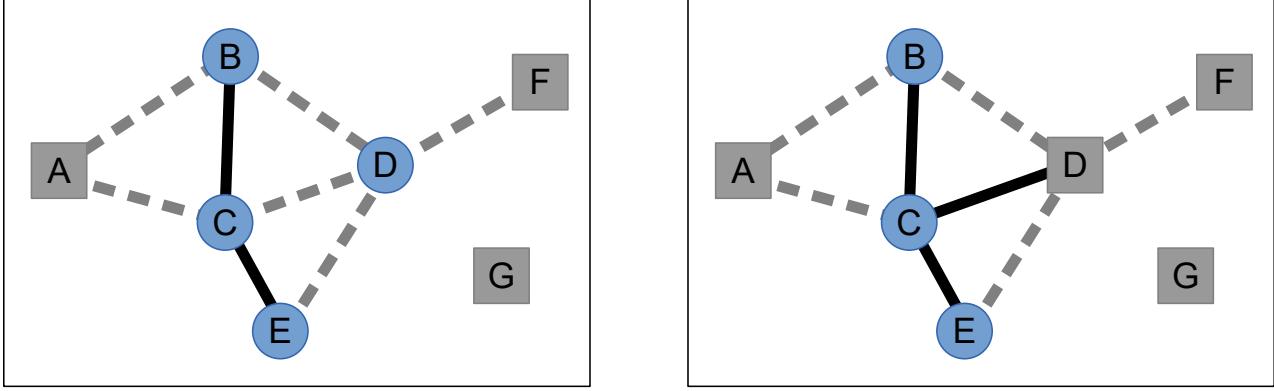


Figure 1.2: On the left one subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ (blue circle nodes and black edges) of the of the network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from Fig. 1.1 is shown. The subgraph has four nodes $\mathcal{V}' = \{B, C, D, E\}$ and two edges $\mathcal{E}' = \{(B, C), (C, D)\}$. A subgraph need not be connected nor need they include all the edges present in the original network given the nodes in the subgraph. For instance the edge between nodes C and D is in \mathcal{E} but has not been included in \mathcal{E}' .

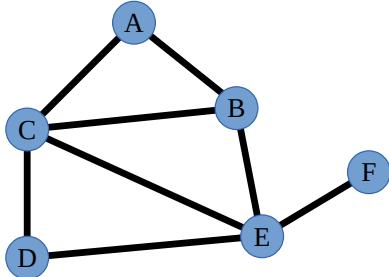
However a subgraph must be a valid network in its own right. Suppose we remove node D from the nodes in the subgraph \mathcal{V}' but add the edge between nodes C and D to the subgraph edge set \mathcal{E}' . This is illustrated in the right-hand picture with blue circles showing the set of nodes $\mathcal{V}'' = \{B, C, E\}$ and the solid black lines showing the edges in $\mathcal{E}'' = \{(B, C), (C, D), (C, E)\}$. This pair $(\mathcal{V}'', \mathcal{E}'')$ no longer forms a subgraph as node D is in not in set of nodes $\mathcal{V}'' = \{B, C, E\}$ but the edge (C, D) is in the set of edges \mathcal{E}'' .

Given the wide range of applications, it should not be surprising to find that networks are often adapted to carry more information. This leads to different more specialised types of network. Most are classified by the type of edge used which describes the information recorded about the bilateral relationships.

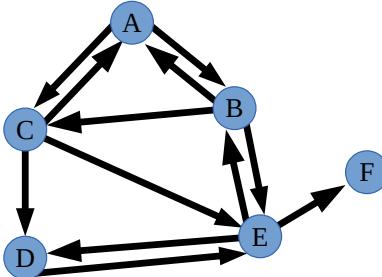
The simplest type of network is, inevitably, called a **SIMPLE NETWORK**. This is a network where there at most one edge between any two pairs of nodes *and* there are no **SELF LOOPS** — edges where both ends are connected to the same node. The network in Fig. 1.1 is a simple network.

A **DIRECTED NETWORK** (sometimes called a **DIGRAPH**) is one where each edge has a direction. We will denote the edge running *from* node u to node v as (u, v) . A node at the start of a directed edge is sometimes called the **SOURCE NODE** (but we also use this term to describe other types of nodes) or **TAIL** while the node at the end of an edge is sometimes called the **TARGET NODE** or the **HEAD**. The natural visualisation of the edge (u, v) , as shown in Fig. 1.4, is to add an arrow pointing away from node u (i.e the tail of the arrow, the source of the edge) and towards node v (so this is the head of the arrow, the target of the edge). Such an edge is distinct from an edge running in the other direction, so (v, u) runs from v to u , see Fig. 1.3b for an example. For instance, if we represent web pages as nodes then the hyperlinks between pages are naturally represented by a directed edge from the node representing the page with the hyperlink is encoded to the node representing the page whose address is in the hyperlink.

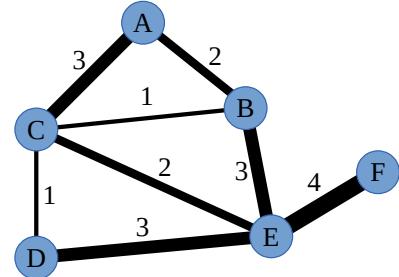
The other useful type of network is the **WEIGHTED NETWORK** in which each edge has a value, the **WEIGHT** (or **EDGE WEIGHT**), see Fig. 1.3c. The value associated with an edge is usually a non-negative number. For instance if the nodes in a network represent distinct phone numbers, then we might add a edge between numbers if there is a phone call between those numbers. We could add an integer weight to each edge to record the number of calls made. Alternatively, we could associate a real number to each edge to record the total time spent on calls between the two numbers.



(a) An example of a simple network. At most one edge between each pair of distinct nodes with no value or direction assigned to the edges. Most networks may be simplified, into a simple network e.g. by ignoring edge directions and weights, removing self-loops. In some cases this is a useful simplification, and sometimes valuable information is lost.



(b) An example of a directed network (or digraph) where the arrow on the edge represents a direction. Note some pairs of nodes are connected in both directions (the relationship is reciprocated), some are just one-way (unreciprocated). The hyperlinks between websites (the nodes) show this type of structure.



(c) An example of weighted network. The values associated with the edges, the **WEIGHTS** are shown beside each edge. For example it could represent the number of calls made in a week between different telephone numbers.

Figure 1.3: Examples of different types of network. The nodes are the same but the edges carry different information. The undirected are strongly connected but the directed network is only weakly connected since there is no path *from* node F to any other node.

Sometimes it may be useful to allow more than one edge between the same pair of nodes. For instance we could use this to represent different phone calls between people occurring at different times.

To represent a network, we need only list the nodes and edges, along with any properties they may have. Such lists are very convenient for numerical work. However for analytical work, and in some cases numerically too, it is useful to represent a network using an **ADJACENCY MATRIX** A . In these notes, the adjacency matrix is defined such that A_{vu} is the weight of the edge (u, v) , that is the edge running from node u to node v , see Fig. 1.4.

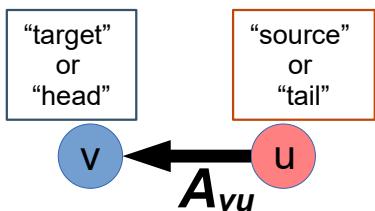


Figure 1.4: The convention used for adjacency matrices in these notes. The node u at the start of the edge is sometimes called the source node or tail . The node v at the end of this edge is sometimes called the target node or the head.

Note that we will sometimes use a numerical label for the nodes where i, j are positive integers running from 1 to N , the number of nodes. Again, A_{ji} represents the edge running from node labelled by integer i to the node indexed by integer j . For an example see Fig. 1.5.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 2 & 3 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 3 & 0 \\ 3 & 1 & 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 \\ 0 & 3 & 2 & 3 & 0 & 4 \\ 0 & 0 & 0 & 0 & 4 & 0 \end{pmatrix}$$

Simple network of Fig. 1.3a. Directed Network of Fig. 1.3b. Weighted Network of Fig. 1.3c.

Figure 1.5: Some example adjacency matrices. The nodes are numbered in alphabetical order so node A has index 1 while node F is index 5. This means that A_{25} represents the edge from node E to node B .

1.2 Basic network measurements

1.2.1 Degree and degree distribution

The DEGREE k_u of a node u is the number of edges attached to that node. So in Fig. 1.3a node C has degree 4. For simple networks

$$k_u = \sum_{v \in \mathcal{V}} A_{uv} = \sum_{v \in \mathcal{V}} A_{vu}. \quad (1.2.1)$$

The degree counts the number of STUBS — the ends of an edge — attached to a node. In social science measures of the importance of a node in a network are known as CENTRALITY measures and degree is perhaps the simplest of these measures. So in Fig. 1.3a nodes 3 and 4 have the highest degree which suggests they might be the most important.

For directed networks, it is necessary to define two types of degree: the OUT-DEGREE $k_u^{(\text{out})}$ is the number of edges leaving node u while the IN-DEGREE $k_u^{(\text{in})}$ is the number of edges pointing into node u . So in Fig. 1.3b node 3 has in-degree 2 but out-degree 3.⁴(T)⁴(T)⁵ In terms of the adjacency matrix we have that

$$k_u^{(\text{in})} = \sum_{v \in \mathcal{V}} A_{uv}, \quad k_v^{(\text{out})} = \sum_{u \in \mathcal{V}} A_{uv}. \quad (1.2.2a)$$

One of the most basic measures of a network is the DEGREE DISTRIBUTION, $n(k)$ which is the number of nodes with degree k (see Jennings (1937) for an early example). This is often phrased as a probability distribution $p(k)$ where

$$p(k) = \frac{1}{N} n(k), \quad N = \sum_{k=0}^{\infty} n(k) = \sum_{v \in \mathcal{V}} 1 \quad (1.2.3)$$

and N is the number of nodes in the network. Another way to say this is that the value of $p(k)$ is the probability of finding a node with degree k when we choose a node at random, or more precisely when we choose a node uniformly at random from the set of nodes⁶. For

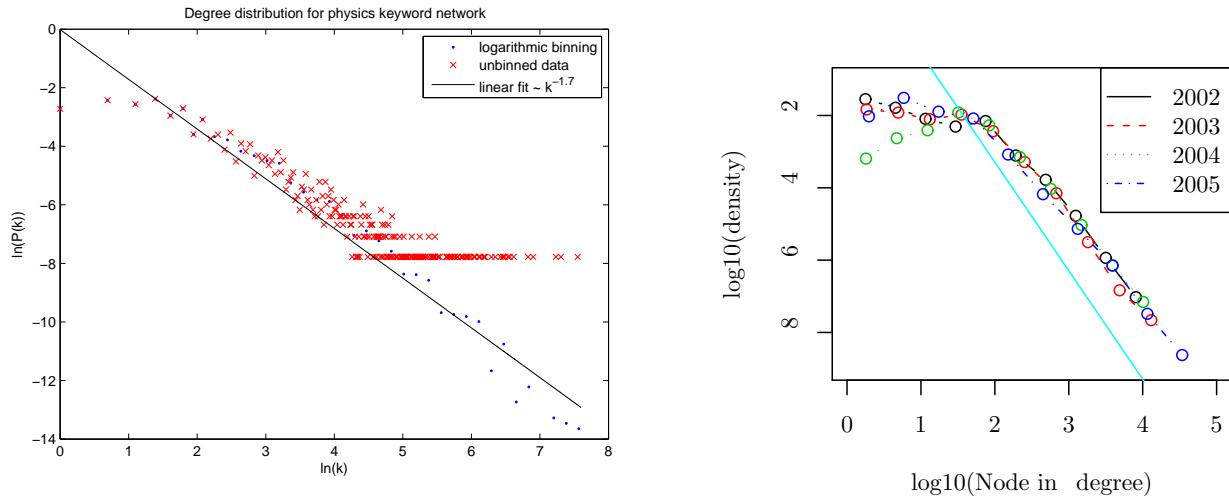
⁴(T)In a directed network, the total degree would be $k_u = k_u^{(\text{in})} + k_u^{(\text{out})}$.

⁵(T)A directed network where $k_u^{(\text{in})} = k_u^{(\text{out})}$ is known as a REGULAR NETWORK.

⁶For this interpretation to work we are formally treating all nodes equally when drawing one of them. However it is common in much of the literature that an unqualified use of the word ‘random’ really means something is being chosen “uniformly at random” from the relevant set. Usually it is pretty obvious what is meant but we will see later that there are situations where we choose nodes with some bias, in particular choosing high degree nodes more often than low degree nodes.

directed networks, we can define separate distributions for the in- and out-degree distributions and these are normally not the same distribution.

One of the features in many real networks is that their degree distribution has a **FAT-TAIL**. That is it falls away for large degree much slower than we would expect from an exponential distribution, such as the binomial degree distribution of the ER network. An example of a fat-tailed distribution is one that falls away as a power-law for large degree, $p(k) \sim k^{-\gamma}$ for large k and some constant power γ . Such distributions are also found in non-network contexts e.g. wealth distributions in economics, word frequency, sizes of earthquakes, and see Figure 1.6a and Figure 1.6b.



(a) The degree distribution of a network of words taken from titles of Imperial Physics publications. Words are linked if they appear in same title (unpublished, Weir & Evans 2006).

(b) The degree distribution from a network describing the way users navigate a photo web site. Nodes are pictures stored on the site and these are linked if a user downloaded these two pictures consecutively (from Argent-Katwala et al., 2007)).

Figure 1.6: Examples of fat-tailed degree distributions from real datasets.

1.2.2 Walks, paths, and cycles

(T)⁷

The degree is a very *local* property of a node, it only counts the number of nearest neighbours and typically does not probe the majority of the network surrounding a node. The main use of network analysis is to enable us to see how the structure on larger scales, beyond nearest neighbours, is important for the system we are studying. One set of ideas which probe this structure come with the concepts of walks and paths.

A WALK \mathcal{W} in a network is a sequence⁸ of nodes, $(v_0, v_1, \dots, v_L) = (v_n)_{n=0}^L$ where there is an edge from each node v_n to the next node in the sequence v_{n+1} so that $(v_n, v_{n+1}) \in \mathcal{E}$ (where $n \in \{0, 1, \dots, (L-1)\}$). For instance in Fig. 1.7 the sequence (C, B, A, B) is a walk. More formally a walk \mathcal{W} satisfies

$$\mathcal{W} = (v_n | v_n \in \mathcal{V}, n \in \{0, 1, 2, \dots, L\}, \text{ and } (v_{m-1}, v_m) \in \mathcal{E}, m \in \{1, 2, \dots, L\}). \quad (1.2.4)$$

⁷(T)Gastner (Grote, 2011) Section 4 page 18. AC (Chamberlain, 2013) lecture 8,9.

⁸We will use round brackets (...) to indicate a sequence to differentiate it from a set, indicated with curly brackets {...}. See appendix A.4 for a discussion of sets and sequences.

It is sometimes convenient to talk about the edges $\mathcal{E}(\mathcal{P})$ in a walk \mathcal{W} . This just refers to the edges traversed when following the sequence of nodes, the edges defined by consecutive nodes in the walk, i.e.

$$\mathcal{E}(\mathcal{W}) = \{(v_{m-1}, v_m)\} \quad \text{where } m \in \{1, 2, \dots, L\} \text{ and } \mathcal{W} = (v_n)_{n=0}^L. \quad (1.2.5)$$

The LENGTH $\ell(\mathcal{W})$ of a walk \mathcal{W} is the number of edges traversed in the walk, the number of nodes in the sequence *minus one*, here⁹ L . For instance in Fig. 1.7 the sequence (C, B, A, B) is a walk of network length 3. Later, at the end of section 1.2.3, we will see there other ways assign a length or weight to a walk. So when we need to avoid confusion, we may talk about the NETWORK LENGTH of a walk when measure this in terms of the number of edges in a walk.

Also worth noting is that this definition includes the TRIVIAL WALK (v) , a walk of length zero containing just one node, which you may want to exclude in some analyses.

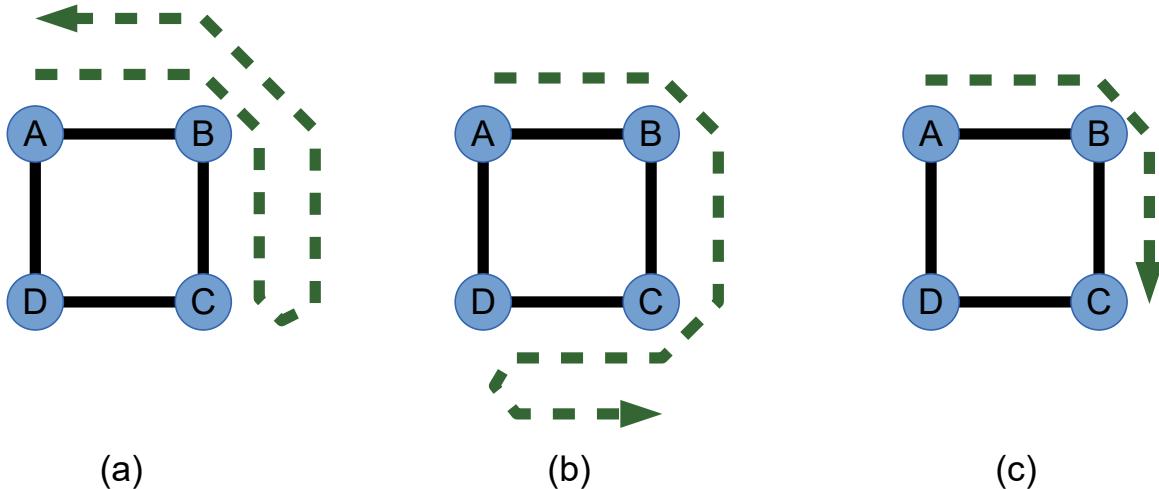


Figure 1.7: On the left, (a) is an example of a walk (A, B, C, B, A) of length 4 and it is also not a cycle. In the centre, (b) is a true cycle (A, B, C, D, A) as no edge is traversed more than once and only one node appears more than once, and that is the initial and final node of the path, here A . A path (A, B, C) of length 2 is shown on the right in (c) where every node in the path appears only once and no edge is traversed more than once.

Walks are important tools when studying how nodes are connected indirectly through the network. In particular, not all nodes may be connected though a path. So a useful concept is reachability where we say that node v is REACHABLE from node u if there is a walk from node u to v .

When a network is directed, the focus in most of this book, you must follow the direction of the arrows along your walk. That is in the definition (1.2.4) above, each neighbouring pair of nodes in the sequence, (v_n, v_{n+1}) , must be present in the edge set \mathcal{E} of the network. If the edges are directed, the presence of an edge in the other direction, (v_{n+1}, v_n) , is not sufficient for the walk \mathcal{W} above to exist. For instance in Fig. 1.3b there the sequence of nodes (C, B, D) is not a walk as there is no edge (C, B) , no edge from node C to node B even though there is an edge in the other direction between this pair of nodes. See Fig. 1.8 for an example. There is an important caveat that not all fields follow this terminology on a directed network for a walk, or the special cases known as a path and a cycle defined below. In fact some fields¹⁰

⁹Note in our definitions of a walk such as (1.2.4) we index the nodes in the sequence starting from zero so the network length is then the index of the last node.

¹⁰For example, a path is a type of walk defined below. It is common in causal inference (see section 5.5) to refer to a CAUSAL PATH when successive node pairs (v_{m-1}, v_m) are an edge. If any of the successive node pairs

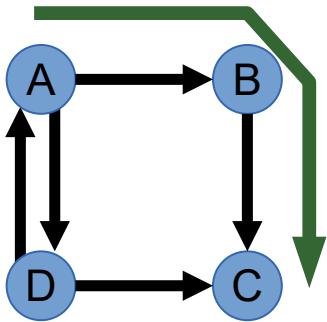


Figure 1.8: The dashed green line shows the directed path (A, B, C) of length 2. However note there is no path in the reverse direction from C to A .

use the term walk (path, cycle, etc) on a directed graph to refer to a sequence of nodes where successive nodes can be an edge in *either* direction so the reader needs to check the definitions of walks and related objects carefully when the material is for a directed network.

Note that a node may appear more than once in a walk¹¹. In particular a CYCLE is a walk in which only the first and last nodes are the same, the rest are distinct, so for a cycle $v_n = v_m$ in (1.2.4) only if $n = m$ or $n = 0$ and $m = L$ or $n = L$ and $m = 0$, see Fig. 1.7.

An network without cycles is called ACYCLIC and this will be an important property the networks we will look at in later chapters. Undirected networks which are acyclic are known as TREES since they like the roots or branches of a TREE, for example see Fig. 1.9. A simple

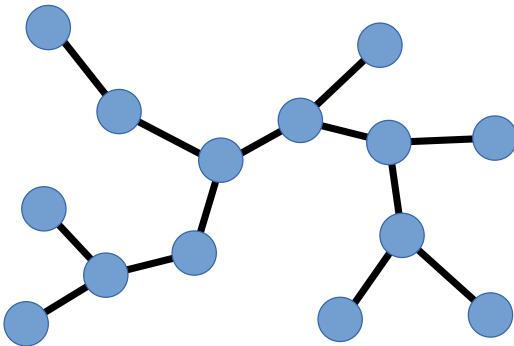


Figure 1.9: An example of a tree, a single component undirected network with no loops. Trees are always examples of planar networks and always have one more node than edges.

test for a acyclic undirected network is to use the formula

$$E - N + C = L \quad (1.2.6)$$

where N is the number of nodes, E is the numbers of edges, and C is the number of components. Here L is the number is *independent* cycles in the network which is a lower bound on the total number of cycles. Cycles can often be seen as the combination (the sum in some sense) of two smaller cycles so L just counts the minimum number of cycles which can be used to create all other cycles in the network. The key result for us is that if there are non independent loops, $L = 0$, the network is acyclic since you cannot build any additional cycles from nothing. So for an undirected network of a single component ($C = 1$) to be acyclic ($L = 0$) we must have that the number edges is one less than the number of nodes. The lack of direction is important and then the formula does not apply to directed networks.

Much of the time, the focus in network analysis is on a special type of walk with *no* cycles, known as a PATH. That is each node in a path $\mathcal{P} = (v_n)_{n=0}^L$ is unique so $v_n = v_m$ iff $n = m$ in (1.2.4). An example is shown in Fig. 1.7. Again, sometimes you may want to include the

(v_{m-1}, v_m) is not an edge but the reciprocal (v_m, v_{m-1}) is an edge, then this is called an ACAUSAL PATH.. The term ‘path’ refers to anything that is a causal or an acausal path. In work on signed networks such as (Harary, 1959, Doreian and Mrvar, 2009) the theorems often refer to ‘cycles’ (another type of walk defined below) but in this context these can be sequences of nodes where successive nodes can be an edge in either direction.

¹¹There is another concept, the TRAIL, which is a walk where no edge appears more than once though a node can appear more than once. We will not use trails in these notes.

TRIVIAL PATHS in your analysis, paths of one vertex which have network length zero, sometimes you may want to exclude such paths.

Finally, it is straightforward to show that if the network is unweighted with only 0 or 1 entries in its adjacency matrix $A_{vu} \in \{0, 1\}$, then number of walks $N_{vu}(L)$ of length L from node u to node v is simply $[\mathbf{A}^L]_{vu}$,

$$N_{vu}(L) = [\mathbf{A}^L]_{vu} \quad \text{if } A_{xy} \in \{0, 1\} \forall x, y \in \mathcal{V}. \quad (1.2.7)$$

To see this we know that every edge from node v to another node w can be added to each of the $N_{vu}(L)$ paths of length L from u to v to create a new path from u to w via v . These new paths from have length $(L + 1)$. If we repeat this for every node v that has an edge ending at w , i.e. where A_{wu} is one, then we have the number of paths arriving at w of length $(L + 1)$. That is

$$N_{wv}(L + 1) = \sum_w A_{wv} N_{wv}(L). \quad (1.2.8)$$

If we suppose that $N_{vu}(L) = [\mathbf{A}^L]_{vu}$ for some L , then we can see from (1.2.8) that $\mathbf{N}(L + 1) = \mathbf{A} \cdot \mathbf{A}^L = \mathbf{A}^{L+1}$. So if the formula is true for any integer L it is true for $(L + 1)$ and so it is true for any integer greater or equal to L . The number of paths of length one arriving at v from u is simply one if A_{vu} is one and zero otherwise. That is $N_{vu}(1) = A_{vu}$. Thus $\mathbf{N}(L) = \mathbf{A}^L$ is true for any positive integer¹² L .

The last feature of walks to highlight is to return to the concept of the “length” of the walk. If we have a weighted network, we can still count the number of edges in a walk and so network length is still a useful concept. However, in some contexts, it is useful to talk about the WEIGHT OF A WALK, $\omega(\mathcal{W})$. **(T)**¹³ This is defined to be the sum of the weights on the edges in the walk, that is

$$\omega(\mathcal{W}) = \omega((v_n)_{n=0}^L) = \sum_{m=1}^L A(v_m, v_{m-1}) \quad (1.2.9)$$

where $A(v, u) \equiv A_{vu}$ is the weight of an edge from u to v . This can only be useful if adding the weights like this has some meaning¹⁴. For instance, if the weight A_{vu} represents the distance between nodes u and v then the sum of weights on the walk represents the distance travelled in moving from the initial v_0 node to final node v_L if you follow the walk. So while we always have the number of edges, the network length as one measure of the “length” $\ell(\mathcal{W})$ of a walk, the path weight gives us an alternative definition of length. And from length, we can start to discuss how to measure distances on a network.

1.2.3 Distance in a network

We have just seen how we can define the “length” $\ell(\mathcal{W})$ of a walk \mathcal{W} by counting the number of edges in a walk. This gives us a ruler for a network and so a way to define the DISTANCE $d(v, u)$ between two nodes u and v . In this section we will focus on the most obvious and most widely used measure of the distance between nodes in a network.

Let us start by thinking about how we lengths and distance. Imagine we have two fixed points in space. We can connect them with a piece of string and measure the length of that

¹²Suppose we allow trivial paths. That is that every node u has a path of length zero from itself to itself given by the sequence of that node alone which we write as (u) . Then $\mathbf{N}(L = 0) = \mathbf{1}\mathbf{1}$. It is consistent to define the zero-th power of a matrix to be the unit matrix. So then we see that $\mathbf{N}(L) = \mathbf{A}^L$ for any non-negative integer, from $L = 0$ upwards.

¹³**(T)**Should we use a different measure $\omega(\mathcal{W})$ for the weight of a walk and keep ℓ for network length?

¹⁴So for unweighted networks the network length $\ell(\mathcal{W})$ is always the same as the path weight $\omega(\mathcal{W})$, that is $\ell(\mathcal{W}) = \omega(\mathcal{W})$.

string. Since string can wind about in space, even loop back on itself, we can find many pieces of string of different lengths, lying in different positions, that connect our two points. However, we will also find that the length of the string cannot go below a certain value. The shortest piece of string that connects the two points is the straight piece. The length of this shortest, straight piece of string is the one which defines what we call the “distance” between our two points.

In the same way, in many network applications, we are interested in the SHORTEST PATH between two nodes¹⁵. For instance, the quickest way to send information between two computers is likely to be via the path with least number of routers lying on the route between our two computers, assuming each router (the nodes of the network) is the bottleneck in that system. We might imagine that the easiest way to arrange a meeting with the President of the United States is to find the path in our social network with the least number of edges so that a friend of a friend of a friend (etc.) can provide the introductions we need.

It is then natural to define the $d^{(sp)}(v, u)$ between two nodes u and v in our network as being the length of the shortest path (i.e. the number of edges in the path) between these two nodes. We will indicate the length of the shortest path from node u to v as $d_{vu}^{(sp)} \equiv d^{(sp)}(v, u)$, following the same convention for direction as used here for the adjacency matrix. To find $d_{vu}^{(sp)}$ we look at all possible paths and find the length of the shortest — there is often more than one path with the same length.

A distance between nodes immediately gives us a very useful characteristic network length scale for a network — the AVERAGE SHORTEST PATH LENGTH $\bar{d}^{(sp)}$. This is the average over all pairs of connected nodes of the network lengths of the shortest path between those pairs. More formally¹⁶

$$\bar{d}^{(sp)} = \frac{1}{N(N-1)} \sum_{u,v \in \mathcal{V} \times \mathcal{V}} d_{vu}^{(sp)}. \quad (1.2.10)$$

It is this measure which is linked to the popular idea about the “six-degrees of separation”, such as used in Guare’s play (Guare, 1990). This is named after the famous experiment by Milgram (1967) in which he showed that letters that reached their destinations when exchanged only between people on first name terms did so after five intermediaries¹⁷.

There are few properties of distances in a network worth noting here. It is often useful to define the distance from a node to itself as zero, $d(v, v) = d_{vv} = 0$. This is consistent with including trivial paths of one vertex v with no edges so such paths have zero length. At the other extreme, it is possible for there to be no path between two nodes u and v , e.g. if they are in different components. The definition in terms of length of shortest paths gives no distance value in this case. Sometimes, the definition of distance is extended to assign a value in this case but this is not always needed and the choice should be made clear. For instance, an infinite value for the distance between unconnected nodes may simplify some expressions for quantities based on the distances between nodes. Finally, in an undirected network, there the shortest path from u to v is also a shortest path from v to u so we have a symmetry in the distance

¹⁵There is no need to consider walks as by definition they contain at least one cycle. Removing all the cycles, i.e. finding the unique path within a walk, can only produce a shorter walk/path between the initial and final nodes, since the length is the number of edges in a walk. So we will focus on paths for the rest of this discussion.

¹⁶Note that this form is standard if every node is connected to every other node. We are assuming that $d(v, v) = 0$ so terms where $u = v$ do not contribute. This means there are $N(N - 1)$ distinct pairs where the u, v and v, u node pairs each contribute once. This latter property means this also works for directed networks since it does not assume that $d(u, v)$ is the same as $d(v, u)$. However, if some node pairs are not connected then this is an example where the distance between disconnected pairs needs to be set to a value. In this context, zero is a more useful value to assign to disconnected nodes rather than infinity. The precise definition used in more complicated cases varies but in practice they all aim to be a measure of the ‘average’ shortest path length.

¹⁷It is well worth reading Watts (2003) on work by Kleinfeld who reviewed the Milgram’s data and methods.

$d(v, u) = d(u, v)$. However in a directed network, there is no guarantee that this is true. If there is a path from u to v and from v to u the shortest paths need not use the same edges and so need not be of the same length, so we may have $d(v, u) \neq d(u, v)$. Further, there need not even be a path in both directions so one of $d(v, u)$ or $d(u, v)$ may have a value representing disconnected nodes. (T)Do we include formal distance definitions here? (end T)

We can be a lot more formal about what we mean by distance. Mathematics has a precise definition of this term and the distance between two points as measured by the network length of the shortest path between those points, $d_{vu}^{(sp)}$, satisfies these criteria. We will look at these criteria later in section 3.3 when we look at various possible measures of distances between the nodes of a DAG.

1.2.4 Components

A COMPONENT of an undirected network is a subgraph in which there is a path between all nodes in the component and *no* path between the nodes of the component and the remaining nodes of the network. Visually components of an undirected network are separate from other components. For instance the network in Fig. 1.10 has three components¹⁸. In terms of set theory, the components define a partition (see section A.4.1) of the set of nodes as every node is in one component and only one component.

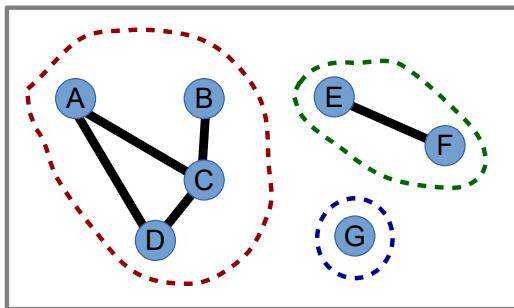


Figure 1.10: A simple network with three components. The network here consists of everything in the box indicated by the grey solid line and contains all seven nodes and the edges between them. There are three components: the subgraph of nodes $\{A, B, C, D\}$ and all their edges, the subgraph of nodes E and F plus the edge between them, and finally the subgraph of the single node G and no edges.

However for a directed network we have a more complicated structure. If we take account of the direction then we say that a component is STRONGLY CONNECTED if there is a path between every pair of nodes in the component. These paths have to take account of the direction of the edges. That means that the path from some node u in a strong component to a node v in the same strong component need not be the same as the path from v to u . If we ignore the direction of the edges we are considering the undirected version of the directed network. The components in this undirected version are referred to as the WEAKLY CONNECTED components(T)¹⁹ of the directed network.(T)²⁰

1.3 Centrality

(T)Local, single path (betweenness, closeness) and all path measures (Katz, PageRank). Examples not a detailed review. (end T)

One of the most common type of measure used in network analysis is called CENTRALITY. We will focus mainly on centrality measures given to nodes but the same ideas can be developed for edges. Typically, node centrality scores are interpreted so that the larger the score the more

¹⁸For another example see the network in Fig. 1.1 which has two components, one the subgraph containing nodes 1 to 6, and the other the network consisting of just the single node 7.

¹⁹(T)Is this true?

²⁰(T)Bow-tie structure of directed networks? Diagram

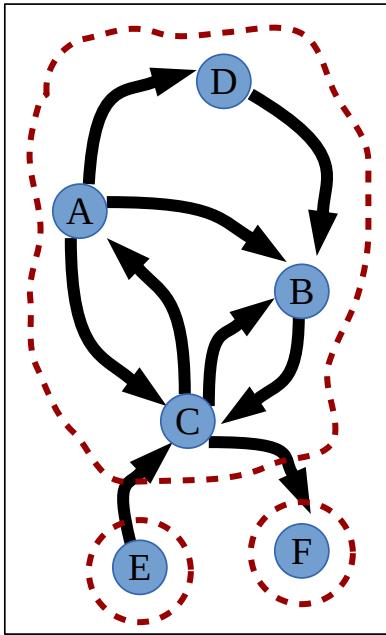


Figure 1.11: The strongly connected components of a directed network. The network here consists of everything in the box indicated by the black solid line and contains all seven nodes and the directed edges between them. Note for instance that any path from node D to node B cannot be the same as any path from D to B because of the direction of the arrows coming in and out of node D. However these two nodes are linked in both directions so are in the same strong component. The three strongly connected components are indicated by the red dashed lines and are the nodes $\{A, B, C, D\}$ plus edges between them, plus the two single node subgraphs based on node E and F respectively. The example network has a single weakly connected component as when we replace directed edges by undirected ones, the resulting simple network has a single component.

important, the more ‘central’, a node is. Here the term centrality is a synonym for importance. The analogy comes from cities where, in many contexts, the political power and wealth creation are concentrated at the centre of the city and so this makes them, in many contexts, the most important parts of a city. However, this analogy also highlights that we may have many different definitions of important which may highlight different parts of the city. Perhaps a modern city has several different centres, a new financial district, an older part where the main political power lies. Perhaps the cultural centre is somewhere else again or the richest and most powerful people do not live in the central areas. Importance, ‘centrality’, in a city can be measured in many ways with different answers. Likewise for network analysis. There are many many ways to measure centrality or importance of nodes (or edges) in a network. Many measures may give similar answers, some will not. Which ever one we choose to use must be a good match for the context of the question we are asking of the network.

The simplest example of a centrality measure is the degree of a node. The number of social connections a person has is often a good indicator of how well connected they are in the social network. However, such a measure is a local measure, that is it is not sensitive to the structure of the whole network.

One way to use the whole network structure to probe the importance of a node is to ask what is the average distance from one node u to all other nodes where distance to some node v is the network distance, the length of the shortest from u to v . The inverse of this is called the CLOSENESS c_u of node u , so it is small when most nodes are far away. In turn high closeness means a node is closer to most nodes, one sign of a central node, an important node. So we can define closeness c_u of node u formally as

$$\frac{1}{c_u} = \frac{1}{N} \sum_{v \in \mathcal{V}} d(u, v) \quad (1.3.1)$$

One set of standard centrality measures is based on the use of the shortest-path between two nodes and a well known example in network analysis is betweenness²¹ (Freeman, 1977).

Another way to use the shortest paths between node to access centrality is to count the number of such paths passing through a node. The idea is that the more important a node is, the more it will be on the optimal route between two other nodes. More formally, if we write

²¹The earliest accessible definition is in Freeman (1977) but see Freeman (1980) for comments.

that the set of shortest paths between source node s and target node t is $\mathcal{P}^{(\text{sp})}(s, t)$, then we may write the node betweenness $b(u)$ of any node u as

$$b(u) = \frac{1}{Z_b} \sum_{s,t \in \mathcal{V}} \sum_{P \in \mathcal{P}^{(\text{sp})}(s,t)} \sum_{v \in P_b \subseteq P} \frac{1}{|\mathcal{P}^{(\text{sp})}(s,t)|} \delta_{u,v}, \quad (1.3.2)$$

where Z_b is some suitable constant normalisation. Here δ_{vu} is the KRONECKER DELTA, simply the entry for the unit matrix so it is one when $v = u$ and is zero otherwise. This formal definition of betweenness is not very helpful but we will look at it in more detail later in section 3.1.2.

One characteristic of closeness and betweenness as centrality measures is that they use a single ‘optimal’ path between two nodes to probe the whole network while assigning importance to a node. A distinctive approach would be to use the many different paths between nodes but to weight them differently. We will look at these types of centrality measure as part of a brief summary of process on a network in the following section.

1.4 Processes on networks

(T)Broadcast, two types of diffusion: PageRank and Laplacian. **(end T)**

One of the main ways to analyse a network is to look at processes moving ‘something’ on the network. These models are often seen as simple, perhaps too simple, models of real process on a network. For example the spread of the rumour in a social network (friends are nodes, their friendships captured by links). In this case we might count how often the rumour reaches a person (node) or how long it takes to reach that person. Another example is when we ask how many people visits a web site. So we need to model the path followed by a user when browsing web pages (nodes of a network) and following hyperlinks (directed edges) to move from page to page. Whether or not the process is a good model of some real process, these can still be useful theoretical tools that probe the whole network structure and help us analyse the network. We can try to use similar processes on a DAG but as we will see later, the special properties of a DAG mean we often have to modify these ideas from their implementation on ordinary networks.

There are many theoretical processes used in the literature but we will focus on a few basic examples that are widely used and which illustrate the basic options available. The first general characteristic of some of these processes is conservation: is quantity moved around the network (copies of a rumour, people browsing web pages) conserved or not. A rumour can be copied endlessly while each person browsing the web is unique. Another characteristic of some processes is whether they access global information or not. That is, at each point in the evolution of the system, does the way that the quantity moves depend on the information nearby, say on the number of neighbours of a node, or does it depend on information about the network as a whole, such as the total number of nodes. For instance, when browsing the web, a user can see all the outgoing links, but has no idea how many web pages there are. Typically real processes will use local information but when designing analysis tools we need not limit ourselves in the same way.

We will use one application of these processes to motivate their use, namely NODE CENTRALITY. That is we will use these processes to create a value or score for each node. These centrality measures use many different paths between nodes to arrive at a score for each node unlike the single optimal path used in the closeness and betweenness measures of the last section.

1.4.1 Broadcasting

This is the simplest process to consider. Suppose we say $w_u(t)$ is the value at node u at time t in the process. This might be the number of times a rumour has been passed to ??? (T) Need a better analogy (end T). The idea is that we pass this value to every neighbour, successors if the network is directed (at the end of out going edges from node u). It is the idea that each node u we share this value with each of its neighbours that leads me to call this a BROADCAST process. When repeated at all nodes simultaneously at time t , adding the quantities that arrive at each node give the new value at each node at the next time step $t + 1$. This simple broadcasting process can therefore be described by the equation

$$w_v(t+1) = \sum_v A_{vu} w_u(t+1). \quad (1.4.1)$$

Since the value at node u at time t is given to every neighbour, the total value $W(t) = \sum_v w_v(t)$ in the network will increase on almost any network.

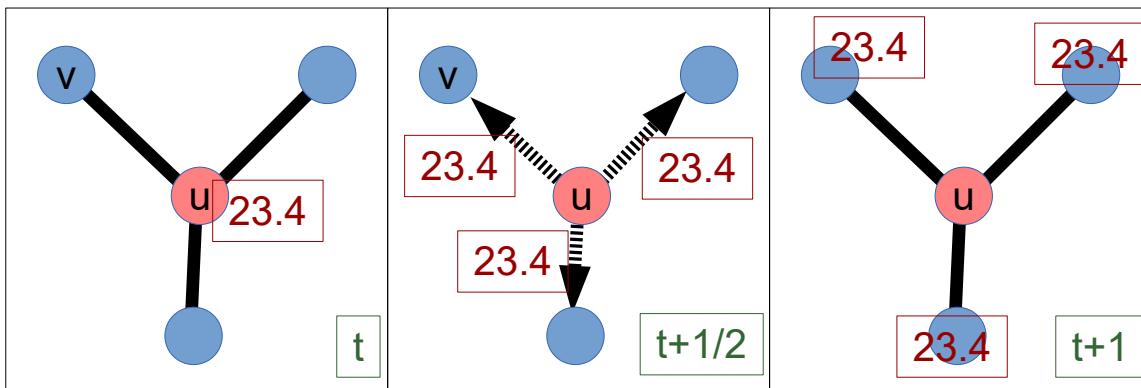


Figure 1.12: Illustration of one iteration of a broadcast process described in (1.4.1) showing how the value from node i alone is distributed. On the left we show that node u starts with value 23.4 at time t . This value is then copied to form part of the value of each neighbouring node at the next time step. There is no amplification or reduction of this value as each edge has weight 1.0. So we can imagine 23.4 flowing along each edge, as shown in the centre at time $(t + 1/2)$, arriving at the neighbouring nodes at time $(t + 1)$. Note that the figure on the right shows that the value of node u at time t does not contribute to the value at u at time $(t + 1)$.

The first such simple process we consider is a BROADCAST PROCESS as shown in Fig. 1.12. In a broadcast process we imagine that at each time step t , every node u has some value $w_u(t)$ representing its importance. As we are considering the context of a centrality measure, this value will be the number of ‘votes’ that node currently has and we are hoping that the number of votes at each node will settle down to some equilibrium which we can interpret. At every tick of the clock, each node v copies or **broadcasts** its current ‘value’ $w_u(t)$ to **all** its neighbours v . Any edge weight A_{vu} for the link from u to v is used as a multiplier, amplifying the signal if the edge weight is greater than one, damping it if the edge weight is less than one. Thus the contribution from node u to the total value at node v at the next time step will be $A_{vu}w_u(t)$. Each node adds up all the values (‘votes’) it receives from its neighbours and this becomes the new value for that node. An important feature of this process is that the number of votes is not conserved as the number of votes sent out by each node is proportional to its strength. Unless the network is regular, unless every node has the same out-strength, the number of votes sent is not just proportional to the w_u at value each node u . The higher degree nodes will have proportionality more influence than lower degree nodes so we are making high degree nodes more important in this broadcast process.

One interpretation of the values $w_v(t)$ for a binary network comes if we set the value of the initial vector at time $t = 0$ to be one for some node s and zero otherwise, $w_s(t = 0) = \text{delta}_{sv}$. We mentioned before that if the edges have weight one or zero, so that $A_{vu} \in \{0, 1\}$, then we have that $[A^n]_{vu}$ is the total number of walks (not paths) of length n from node u to node v . So for such binary networks (such as a simple network) the broadcast process values $w_v(t)$ are proportional to the number of long walks passing through that node v which started at any node where $w_v(t = 0)$ was one.

There are several variations of the broadcast process that are often seen, such as $w_v(t + 1) = h_v + \sum_u \alpha A_{vu} w_u(t)$ for some fixed vector \mathbf{h} and parameter α which can be analysed with linear algebra yet give us more flexibility. These variants are exploited in the definition of centrality measures, see section 1.3.

1.4.2 Diffusion

An alternative to the broadcast process is a DIFFUSION PROCESS on the network. In this case, each node u divides its current ‘value’ between its neighbours v in proportion to the weights of the edges leaving node u , i.e. in proportion to A_{uv} . Thus the flow of ‘votes’ away from a node is divided in proportion to edge weights leaving that node as illustrated in Fig. 1.13. This means the number of ‘votes’ on the network is conserved.

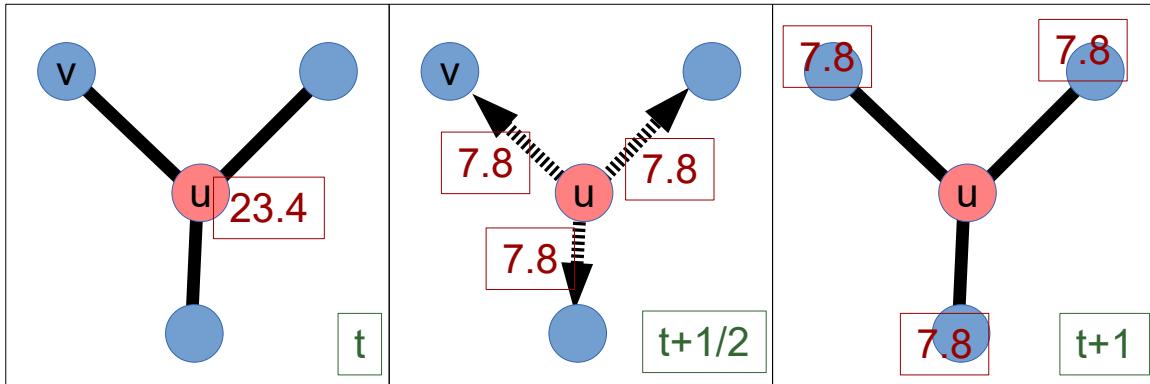


Figure 1.13: Illustration of one iteration of a diffusion process showing how the value from node u alone is distributed. On the left we show that node u has value 23.4. This value is then split equally between the three neighbours (all edges have equal weight) so a value of 7.8 is sent to form part of the value of each neighbouring node at the next time step. We can imagine 7.8 flowing along each edge, as shown in the centre, to arrive at the neighbouring nodes at time $(t+1)$. Note that on the right, the value at node u at time t does not contribute to its own value at time $(t+1)$ as there is no self-loop at u .

The diffusion process is described by the TRANSFER MATRIX $T^{(\text{out})}$, where $T_{vu}^{(\text{out})}$ describes the fraction of the value from source node v passed along one edge to a target neighbour node u . Then if $w_v(t)$ is the number of random walkers at node v at time t we have that

$$w_v(t + 1) = \sum_u T_{vu}^{(\text{out})} w_u(t) \quad (1.4.2)$$

where

$$T_{vu}^{(\text{out})} = \frac{1}{s_u^{(\text{out})}} A_{vu} \quad s_u^{(\text{out})} = \sum_v A_{vu} \quad (= k^{(\text{out})} \text{ if } A_{ij} \in \{0, 1\}). \quad (1.4.3)$$

Example 1.4.1. Transfer Matrix for a Simple Network

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \quad \mathbf{T}^{(\text{out})} = \begin{pmatrix} 0 & 1/2 & 1/4 & 1/3 & 0 \\ 1/3 & 0 & 1/4 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 1/4 & 0 & 1/2 \\ 0 & 0 & 1/4 & 1/3 & 0 \end{pmatrix} \quad (1.4.4)$$

n	1	2	3	4	5
λ	1.00	0.27	-0.17	-0.61	-0.50
$\mathbf{v}_1^{(n)}$	-0.46	-0.45	-0.53	-0.62	-0.00
$\mathbf{v}_2^{(n)}$	-0.31	-0.55	0.27	0.34	0.41
$\mathbf{v}_3^{(n)}$	-0.62	-0.00	0.53	0.00	-0.82
$\mathbf{v}_4^{(n)}$	-0.46	0.45	-0.53	0.62	0.00
$\mathbf{v}_5^{(n)}$	-0.31	0.55	0.27	-0.34	0.41

The transfer matrix for the diffusion process also describes the probability that a “random walker” at source node u moves to target node v at the next time step. The value $w_u(t)$ is then the expected number of random walkers found at node u at time t .

One important feature of this diffusion process is that the random walker must go somewhere and we don’t add any more walkers, so the total number of random walkers is conserved. Equivalently we noted before that in diffusion the number of votes on the network is conserved. This conservation law is reflected in the property that the sum of each column of \mathbf{T} must equal 1, e.g. the sum of the u -th column is

$$\sum_v T_{vu}^+ = \sum_v \frac{1}{s_u^{(\text{out})}} A_{vu} = \frac{1}{s_u^{(\text{out})}} \left(\sum_v A_{vu} \right) = 1 \quad \text{if } s_u^{(\text{out})} > 0. \quad (1.4.5)$$

This is a property of matrices known as STOCHASTIC MATRICES. If we now look at the total number of random walkers, $W(t) = \sum_u w_u(t)$ we see that by using (1.4.5) we have that

$$W(t+1) = \sum_v \sum_u T_{vu}^+ w_u(t) = \sum_u \sum_v T_{vu}^+ w_u(t) = \sum_u 1 \cdot w_u(t) W(t). \quad (1.4.6)$$

That is the number of random walkers is preserved. This all assumes that $s_u^{(\text{out})}$ is non-zero for all nodes. That is if the network is directed we cannot have a node with no out-going edges, so no sink nodes. These would be dead ends for the random walkers and we would need to modify our equations to deal with this case, something that is often done in practice when using this diffusion process as the basis for a network analysis tool.

1.4.3 The Laplacian

We can also define a second type of conserved process but this is most naturally described in terms of a continuous time process. In this case we think of values $w_u(t)$ of leaving at a rate proportional to the strength of every edge leaving node v . Of course, we also have the corresponding process where new material is diffusing along edges coming into the node v . So we have

$$\frac{\partial w_v(t)}{\partial t} = -\lambda \underbrace{\sum_u A_{uv} \mathbf{w}_v(t)}_{\text{flow out}} + \lambda \underbrace{\sum_w A_{vw} \mathbf{w}_w(t)}_{\text{flow in}}, \quad (1.4.7)$$

where λ is some fixed rate parameter. We can rewrite this in terms of a well known operator called the GRAPH LAPLACIAN, or often simply the LAPLACIAN which is the matrix \mathbf{L} where

$$L_{vu} = s_v^{(\text{out})} \delta_{vu} - A_{vu} \quad (1.4.8)$$

so the evolution equation is then

$$\frac{\partial \mathbf{w}(t)}{\partial t} = -\lambda \mathbf{L} \mathbf{w}(t). \quad (1.4.9)$$

Here δ_{vu} is the KRONECKER DELTA, simply the entry for the unit matrix so it is one when $v = u$ and is zero otherwise. The Laplacian usually encountered in the form for simple networks where

$$L_{vu} = k_v \delta_{vu} - A_{vu} \quad (1.4.10)$$

where we have the degree k_v instead of the out-strength.

The main difference from our first diffusion example (1.4.2) is that this diffusion equation is controlled by the adjacency matrix \mathbf{A} and not the stochastic matrix \mathbf{T} of (1.4.5). However, the total amount at the nodes $W(t) = \sum_u w_u(t)$ is still conserved unlike the broad cast process of (1.4.1). This is some sense a ‘conserved broadcast’ process. To see conservation we simply sum (1.4.7) over all nodes to find

$$\sum_v \frac{\partial w_v(t)}{\partial t} = \frac{\partial W(t)}{\partial t} = -\lambda \sum_u \sum_v A_{uv} \mathbf{w}_v(t) + \lambda \sum_v \sum_w A_{vw} \mathbf{w}_w(t) \quad (1.4.11)$$

$$= -\lambda \sum_v s_v^{(\text{out})} \mathbf{w}_v(t) + \lambda \sum_w s_w^{(\text{out})} \mathbf{w}_w(t) = 0. \quad (1.4.12)$$

Note that this is true regardless of the values of the out-strength. In this case, if some nodes are sink node with no outgoing edges, then what will happen is the value at such sink nodes will only ever increase as it can never leave those nodes. So after an infinite amount of time, all the value in the network will end up at the sink nodes.

Finally, we could easily replace the time derivative by $\partial W(t)/\partial t$ in (1.4.12) by the finite time step approximation $(w(t + \Delta) - w(t))/\Delta$. This finite time form is not exactly the same process but it is a valid variant in its own right and has many of the same properties of the continuous time form.

1.5 Classic network models

There are many models which can produce different types of networks. We will look at the two most important models. Both are stochastic so really they define a collection of networks rather than one. These models are rarely realistic but they are extremely useful as NULL MODELS used to make comparisons against networks found in real data. Suppose we have a network created from a data set or a theoretical model and we make some measurements on this network: the degree distribution, the number of components or whatever. How do we assess the meaning of the values we obtain? How do we know if a degree of a node is unusually large? How do we know if the average path length is short? The way to make these statements is to make a comparison with some appropriate standard, what are known as null models. (T)²² That is we make measurements on the network of real interest, one coming from data or a sophisticated model, and compare this against measurements made on our null model.

1.5.1 Erdős-Réyni random network

(T)²³

Around 1959, Erdős and Réyni produced a number of ground breaking results (Erdős and Réyni, 1959) for what is the simplest model of a COMPLEX NETWORK, i.e. something which

²²(T) See L10 of (Clauset, 2013).

²³(T) L10.2 Clauset notes (Clauset, 2013).

is not a regular lattice. However some of these results had also been obtained earlier, by Solomonoff and Rapoport in 1951 (Solomonoff and Rapoport, 1951) and also appeared independently in another 1959 paper (Gilbert, 1959),^{(T)²⁴} but the model is usually linked to the names of Erdős and Rényi. Such complex networks are usually much more important when a network has been created by many small parts interacting without access to detailed global information about the structure they are creating. For instance in the wiring of the brain and other biological systems, physical systems created by many interacting groups of humans such as the hardware underlying the internet, and social networks such as friendship networks.

^{(T)²⁵}^{(T)²⁶}

ASIDE Paul Erdős (1913–1996)



Paul Erdős was a famous itinerant mathematician who produced a prodigious number of papers with a vast number of different co-authors, at least in terms of the norms of mathematics. As a result mathematicians like to play a game where they calculate their Erdős number. This is the shortest path from an author to Erdős in the CO-AUTHORSHIP NETWORK, that is a network with researchers as the nodes linked if they have co-authored a paper. My Erdős number is at most 4 with a path to Paul Erdős via James Clough, Xiang Li and Guantao Chen. Figure from Wikipedia Commons.

The model creates simple networks in one of two ways:

- A. N nodes in which each pair of nodes is connected with probability p , independent of all other pairs,
- B. N nodes with E nodes connected chosen at random with uniform probability from $N(N - 1)/2$ pairs.

As both definitions are stochastic, it means each defines an *ensemble* of networks, that is a set $\mathcal{G} = \{G_1, G_2, \dots\}$ of individual simple networks G_i , not a single network. The difference is that in the set created with definition (A) the number of edges varies between different networks while the networks created using definition (B) all have the same number of edges. So we have $\mathcal{G}_B \subseteq \mathcal{G}_A$. The difference between the two definitions is just the same as the difference between the canonical and microcanonical ensembles in statistical mechanics. Just as with these statistical physics ensembles, there is no difference between the two random network models in the limit of infinite size $N \rightarrow \infty$.

Most precise mathematical results are derived in this infinite network limit and so mathematicians work with the definition which is easiest mathematically, definition A. Computationally, method B is better because only E random numbers are considered (choosing one of N nodes) whereas method A requires a random number for all T possible node pairs, so $T = N(N - 1)/2$. The differences between the two models are likely to be small and of little practical interest unless N is very small. Which ever way we approach our ensembles, an ensemble of Erdős-Rényi random networks make the the simplest possible null models. A typical use of these models is to construct the Erdős-Rényi random networks to have the same number

²⁴^(T)Check this claim.

²⁵^(T)Note the family name of Erdős' has a double acute accent on the o, ö (unicode U + 0151, ő in HTML, \H{o} in LATEX.), and is not a simple umlaut, ö.

²⁶^(T)1959 (see [Bollobas (2001)] for detailed exposure) from Lambiotte & Masuda (Masuda and Lambiotte, 2016) chapter 3

of nodes and edges as the network of interest to us, and we compare the measurements we make. We will work with definition A in this text.

Since these models create an ensemble of networks, the measurements we make should be defined to be averages over these random networks. So, if \mathcal{G}_A is the set of networks created under definition A, we are interested in

$$\langle Q \rangle_A = \frac{\sum_{g \in \mathcal{G}_A} Q(g)}{\sum_{g \in \mathcal{G}_A} 1} \quad (1.5.1)$$

where $Q(g)$ is the value of some quantity A measured on network g .

For definition A the degree distribution is simply a binomial distribution

$$p(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}, \quad (1.5.2)$$

since we have $N - 1$ as the model does not allow self-loops. That is because every edge is created independently of any other edge. So if we look at one node it has $(N - 1)$ potential edges. Each one is present with probability p and not present with probability $(1 - p)$. If k particular edges are present that occurs with probability p^k while the remaining $(N - 1 - k)$ edges are not present with probability $(1 - p)^{N-1-k}$. As it does not matter which k edges are present there are $\binom{N-1}{k}$ different ways we can choose the k edges which are present. The final point is that we could do this for any node, there would be no difference in the argument. So this distribution is also what we would measure if we asked what is the degree of a node chosen uniformly at random from the set of nodes, the probability distribution $p(k)$.

From the properties of a binomial distribution we deduce that that the average degree is $\langle k \rangle = p(N - 1)$. If we work from our definition in terms of an average over the ensemble of networks as have that

$$\begin{aligned} \langle k \rangle_A &= \frac{1}{\sum_{g \in \mathcal{G}_A} 1} \sum_{g \in \mathcal{G}_A} \frac{1}{N} \sum_{i \in \mathcal{V}_k} k_i \\ &= \sum_k p(k)k = p(N - 1) \end{aligned} \quad (1.5.3)$$

Note that we use the fact that there is no difference between nodes in this model, so that on average they will all have the same properties so each term in the sum over individual nodes gives the same answer. The binomial degree distribution encode the result of measuring degree over the ensemble A for any N so that is how we deal with the sum over networks.

As there are N nodes the average degree tells us that in this ensemble A of random networks the number of edges is simply $\langle E \rangle = pN(N - 1)/2$. Thus when want to use this model to make a comparison with a second network of N_2 nodes and average degree $\langle k \rangle_2$ edges, we would chose the parameters of our random network ensemble A to be choose $N = N_2$ and $p = \langle k \rangle_2/(N_2 - 1)$.

It is useful to remember that in the limit $N \rightarrow \infty$ with $\lambda = pN = \langle k \rangle$ held constant, the binomial distributions becomes the Poisson distribution

$$p(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \text{ with } \lambda = pN = \langle k \rangle \quad (1.5.4)$$

So for this large network limit we have that the standard deviation for the average degree is just $\langle k \rangle/\sqrt{N}$. That is the ensemble A of random networks becomes increasingly like a **REGULAR NETWORK**, a network where all nodes have the same degree. For the same reason the total number of edges becomes almost always equal to $E = \langle k \rangle N/2 = pN^2/2$. Thus when these random network models become infinite ($N \rightarrow \infty$) the two ensembles A and B become identical²⁷.

²⁷This is true for exactly the same reasons that the canonical and microcanonical ensembles in statistical mechanics become identical in the thermodynamic limit.

1.5.2 Configuration model

We have mentioned that fat-tailed degree distributions are common, as seen in Fig. 1.6. However, Erdős-Rényi random networks with their binomial degree distribution have an exponential cutoff. So for networks with a fat-tailed distribution we need a better null model, that is a model against which we can compare when investigating network problems. The CONFIGURATION MODEL allows us to specify any degree distribution we want. Typically we would take the same degree distribution as the network we are investigating.

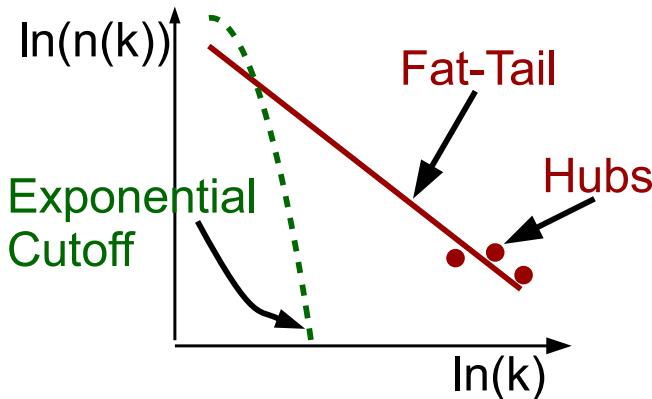


Figure 1.14: Sketch of two possible degree distributions for same total number of nodes and edges. The green dashed line shows an exponential tail, such as an ER network. The red solid line shows a distribution with a power-law tail. The lines represent a typical result averaged over many realisations for large N in an appropriate model. In practice for finite N , each realisation of a model with a power law distribution has a few nodes with very large degree k which are known as hubs, as indicated on the diagram.

The CONFIGURATION MODEL is a random network where the degree of each node $\{k_i\}$ is given. This was discussed by Molloy and Reed (Molloy and Reed, 1995) but the model has been discussed at least as far back as 1972 see discussion and references in (Dorogovtsev et al., 2003) (specifically Békéssy et al. (1972), Bender and Canfield (1978), Bollobás (1980), Wormald (1981)).^(T)²⁸ There are two variations, each of which leads to a slightly different ensemble of networks.

A - $\mathcal{G}(N, p(k))$ N nodes in which each pair of nodes (i, j) is connected with probability $p(i, j) = \tilde{k}_i \tilde{k}_j / (2E)^2$, independent of all other pairs,

B - $\mathcal{G}(N, \{k_i\})$ N nodes, node u has degree $k_i = \tilde{k}_i$ exactly.

These are stochastic definitions defining ensembles of networks $\mathcal{G} = \{G_1, G_2, \dots\}$. Definition **A**, $\mathcal{G}(N, p(k))$, is typically best for mathematical work. Definition **B**, $\mathcal{G}(N, \{k_i\})$ is easier to implement numerically. The definitions equivalent in the large network limit $N \rightarrow \infty$ c.f. canonical and microcanonical ensembles. If degree distribution is Binomial then we get an ER random network.

In our definition we stated that the configuration model was a ‘random network’. What this means is that these random networks have **no statistical correlations** between neighbouring nodes. That is in our ensemble of networks, the only thing which remains constant over all ensembles is the degree of each node. The relationship between nodes, the edges, is randomised.

One numerical implementation starts from a given degree distribution $\{k_i\}$ and the set of nodes (Newman et al., 2001). Definition **B**, $\mathcal{G}(N, \{k_i\})$, with list of $\{k_i\}$ given.

1. Assign k_i stubs (half an edge) to each node u .
2. Choose two stubs at random and join them into an edge.
3. Repeat until all stubs joined.

²⁸(T)A. Bekessy, P. Bekessy, J. Komlos, Studia Sci. Math. Hungar. 7 (1972) 343; E.A. Bender, E.R. Canfield, J. Combin. Theory A 24 (1978) 296; B. Bollobás, Eur. J. Combin. 1 (1980) 311; N.C. Wormald, J. Combin. Theory B 31 (1981) 156,168.

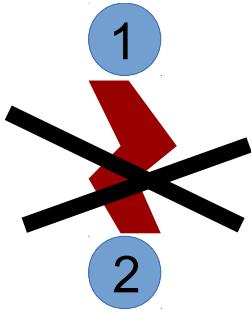
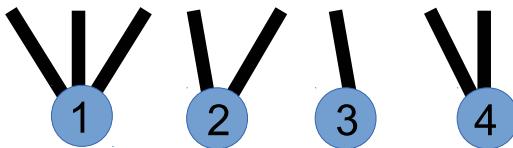
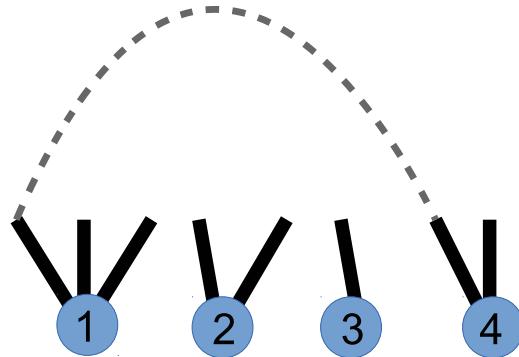


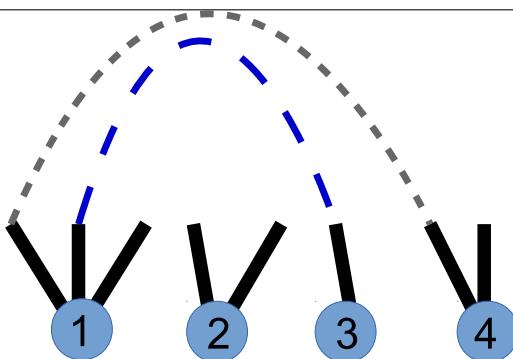
Figure 1.15: A typical assumption for the simplest analytical solutions is that there are no node correlations. That is the properties of one node are not related to the properties of any of its neighbours.



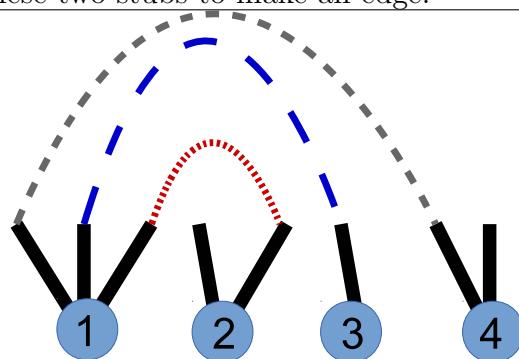
Stage 1: Specify the degree k_i for each node u and give each k_i stubs (half an edge)



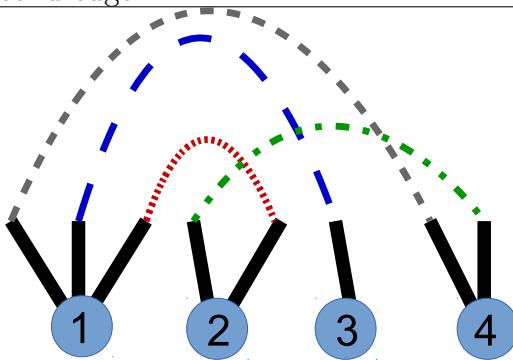
Stage 2: Pick two distinct stubs uniformly at random from the set of stubs. Connect these two stubs to make an edge.



Stage 3: Pick more two distinct stubs uniformly at random from the set of remaining stubs. Connect these two stubs to make a second edge.



Stage 4: Repeat this process for remaining stubs.



Stage 5: Finish when all stubs are paired off as edges.

Figure 1.16: Numerical implementation of configuration model from a list of degrees

In an alternative numerical implementation (Maslov and Sneppen, 2002) we start from an actual network. Definition **B**, $\mathcal{G}(N, \{k_i\})$, with existing network given e.g a data set

1. Choose two edges at random.
2. Swap one end of each edge with each other.
3. Repeat until every edge visited once.

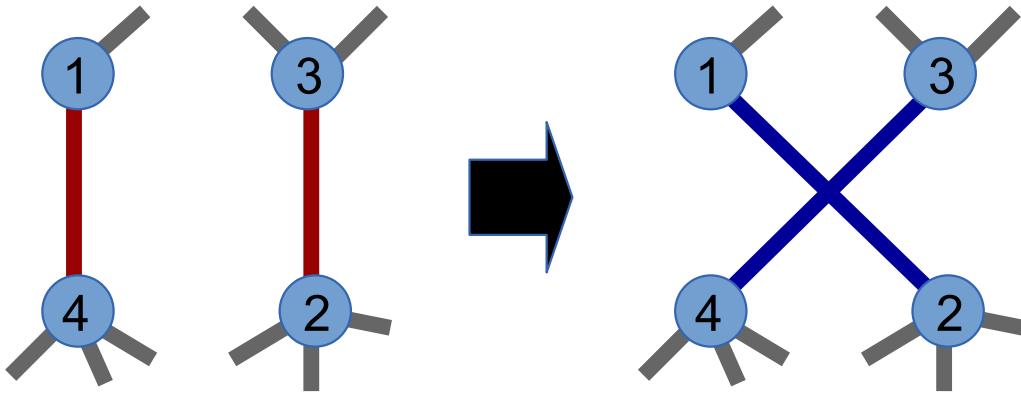


Figure 1.17: Illustration of how to rewire a pair of edges which preserves the degree of nodes.

Chapter 2

DAG Properties

(T)¹

Aims and Objectives: To give define a DAG and to define key properties. Relationship to other formal mathematical objects.

- Definition of Network, Simple Network and Directed Network
- Paths, Shortest Path, Diameter, cycles
- Connectivity & Components, Strong and Weak

2.1 What is a DAG?

A DIRECTED ACYCLIC GRAPH, a **DAG**, $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, is a directed graph with no cycles. That is if you move from one node to any other node connected to it, respecting the direction of the arrows (a walk), then you can never come back to your starting point². This restriction, the lack of any loops in the network, looks relatively simple yet turns out to be an extremely potent constraint on a network. It means that there is an inherent order in the nodes. It acts like an “arrow of time” which reflects the fact that we can never go back in time to change history. It is like the order enforced on natural river basins, where the water can only ever flow down towards the sea.

Many systems show this order, this arrow of time, where it is expressed in different ways.

- Citation Networks. If nodes are documents, their publication gives an order to the nodes. Documents will list only older documents in their bibliographies, relationships which we use to define the edges in a citation network..
- Computer Package Dependency Networks. Every package is a node and the edges are directed from one package to another that requires it. There can never be a cycle in the way packages depend on other packages.
- Spreadsheets. We can let each cell be a node of a network. If a cell has a formula we can make links from the cells contain values required into the cell with the formula. The logic in the formulae in a spreadsheet means that there cannot be a cycle in the way one cell takes values from another, otherwise an error is given.
- Mathematical Proofs. For the same reason as the formulae in a spreadsheet, any mathematical proof cannot contain “circular arguments”. (T)³

¹(T)Friday, January 4, 2019 13:05

²So any walk on a DAG is always a path.

³(T)Nodes and edges here?

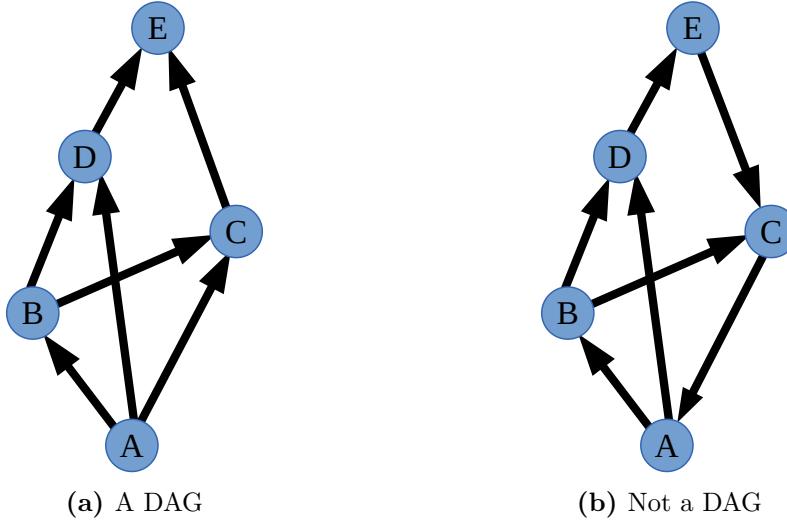


Figure 2.1: Two example networks. On the left is a DAG where the order is clear in this visualisation since all the edges point upwards, never downwards. On the right is a directed network in which is identical to the network on the left except the direction of two edges has been changed: (A, C) and (C, E) have been reversed. We can now see several cycles in the right hand network, e.g. $A \rightarrow B \rightarrow C \rightarrow A$, and this directed network not acyclic. Indeed, the network on the right is a strongly connected directed network, there is a path from every node to every other node.

- Food Webs. In a food web the species are the nodes with a connection from a predator to a species it preys on. Food webs encode the natural order within predator and prey relationships, the flow of energy in the system.
- Bayesian networks
- GRAPHICAL CAUSAL MODELS (Pearl, 1995) which are “the interpretation of DAGs as nonparametric structural equation models (NPSEM) unquestionably dominates the literature” (Elwert, 2013) and also see (Pearl, 1995, Greenland et al., 1999, Rohrer, 2018). From (Rohrer, 2018) DAGs “were developed primarily by the computer scientist Judea Pearl (e.g., Pearl, 1995; see Pearl, Glymour, & Jewell, 2016, for an introduction) and share many features with structural equation models (SEMs)”.
- Power Grids where the DAG represents the instantaneous state of a power system (Ugwu and Cuffe, 2023).
- **(T)**Other examples or do we have enough? **(end T)**

We will look at some of these examples in more detail in chapter 5.

Some simple
illustrations of real
world DAGs.

Figure 2.2: **(T)**Some simple illustrations of real world DAGs **(end T)**.

So systems with an inherent order, with no cycles, are common and, as we shall see, the pattern of the relationships, the topology of the network, acquires special properties because of this order.

The food web though provides a nice counter example. There is, of course, ultimately some sort of cycle in the energy flow encoded in a food web if its is to be stable. The top predators die and part of the energy stored in their body mass is recycled by the smallest parts of the

ecosystem, such as bacteria, at the bottom of the ‘food chain’. So most food webs are only DAGs if we look at a small part of the ecosystem. Even then we may have to ignore parasitic behaviour which are another source of cycles in a food web. That illustrates a common feature with real data sets that for all sorts of reason, the real world may have a few cycles encoded in the data. Those few cycles then prompt another question. Is there a systematic way to remove such cycles to produce a true directed acyclic graph? In some cases these cycles are due to errors or special cases of no interest, for instance as happens in citation data. In other cases these cycles may include real links but they are not relevant for the analysis we wish to do, as happens with links from small to large organisms in food web networks. We will look at that in section 6.6.

(T)Directed Trees. ([end T](#))

(T)Reachability, \preceq and \succeq notation. ([end T](#))

The lack of cycles in a DAG means that if there is a path between two nodes⁴, it can only be in one direction. The path will pass through nodes respecting the order inherent in a DAG. To reflect this directionality, if there is a path from a node u to a node v in the DAG we will use the notation $u \preceq v$. This includes the trivial path case, where $u = v$ and the path is simply the sequence (u) of one node, that is $u \preceq u$ which is true for all nodes. If we write $u \prec v$ then we are excluding the trivial path case, so $u \not\sim u$. In general, for any two *distinct* nodes u, v we have three possibilities: $u \prec v$, $v \prec u$ or there is no path between the two nodes. To summarise

$u \preceq v \Leftrightarrow$	path from u to v including case $u = v$	(2.1.1a)
$u \prec v \Leftrightarrow$	path from u to v excluding case $u = v$	(2.1.1b)
$u \succeq v \Leftrightarrow$	path to u from v including case $u = v$	(2.1.1c)
$u \succ v \Leftrightarrow$	path to u from v excluding case $u = v$	(2.1.1d)

2.2 Local Measures

The simplest property of a node in a DAG is to count the number of incoming and outgoing edges, the IN-DEGREE $k_v^{(\text{in})}$ and the OUT-DEGREE $k_v^{(\text{out})}$ of a node v respectively. These are exactly as in any directed network.

Sometimes it is useful to refer to the set of nodes which are the nearest neighbours to a given node v . We will call those neighbours which are the sources of edges (T)⁵ arriving at a node v to be the PREDECESSORS⁶ of v and the set of predecessors is denoted as $\mathcal{N}^-(v) \equiv \mathcal{N}_v^-$. Likewise, the SUCCESSOR⁷ nodes of v are the target nodes of edges which leave node v and we denote this set as $\mathcal{N}^+(v) \equiv \mathcal{N}_v^+$. See Fig. 2.3 for an example. More formally

$$\text{predecessors} \quad \mathcal{N}_v^- = \{u | (u, v) \in \mathcal{E}\}, \quad (2.2.1a)$$

$$\text{successors} \quad \mathcal{N}_v^+ = \{u | (v, u) \in \mathcal{E}\}. \quad (2.2.1b)$$

Thus the in-degree of a node v is just the size of the predecessor set of that node, and the out-degree is the size of the set of successors

$$\text{in-degree} \quad k_v^{(\text{in})} = |\mathcal{N}^-(v)|, \quad (2.2.2a)$$

$$\text{out-degree} \quad k_v^{(\text{out})} = |\mathcal{N}^+(v)|. \quad (2.2.2b)$$

⁴In any network, if there is a path from node u to node v we say v is REACHABLE from u .

⁵(T)Should we avoid using terms sources and targets for edges?

⁶The predecessor nodes are also sometimes called the PARENTS of node v . The set of predecessors is sometimes called the PAST NEIGHBOURHOOD of node v .

⁷The successor nodes are also sometimes called the DESCENDENTS or CHILDREN of node v . The set of successors is sometimes called the FUTURE NEIGHBOURHOOD of node v .

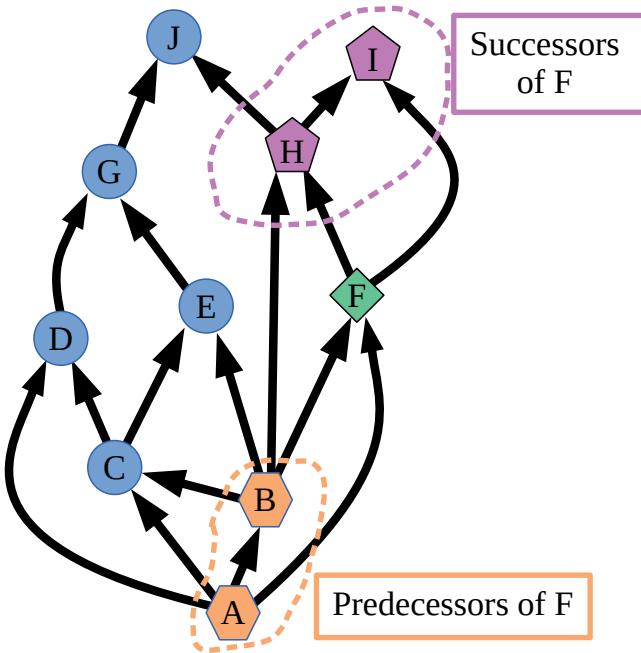


Figure 2.3: An illustration of the past neighbourhood and the future neighbourhood of a node. Here node F (green diamond) has two predecessors, the neighbours in the past nodes A and B (orange hexagons), and two successors, the neighbours in the future nodes H and I (purple pentagons). Note node J is *not* a successor as though it is in the future of F there is no edge from F to J . The only node with zero in-degree is node A so this is the only source node for this DAG. However both node I and J have no out-degree so these are the two sink nodes for this DAG. We can also see that the ancestors of F are in this case the same as the predecessors but the descendants of F the two child nodes H and I together with node J . **(T)**THIS IS THE CORRECT LABELLING for standard 10 node DAG example. WATCH OUT FOR D AND F LABEL SWITCH. **(end T)**

Since we have an arrow of time we can have a definitive start and finish to our DAG. The starting points of the DAG are known as **SOURCE NODES**⁸ which are ones with zero in-degree. There is always at least one of these but there can be many. In the same way nodes with zero out-degree are the **SINK NODES** which we can think of as being at the end of the DAG. We will denote the set of all source nodes as \mathcal{S} while the set of all sink nodes is \mathcal{T} (t stands for ‘target’ here), or more formally

$$\text{source nodes} \quad \mathcal{S} = \{s | k^{(\text{in})}(s) = 0, s \in \mathcal{V}\}, \quad (2.2.3a)$$

$$\text{sink nodes} \quad \mathcal{T} = \{t | k^{(\text{out})}(t) = 0, t \in \mathcal{V}\}. \quad (2.2.3b)$$

See Fig. 2.3 for an example.

We can look at sources and sinks in another way. DAGs have an order which implies that finite sized DAGs must have a *beginning* (sources) and an *end* (sinks). Since they have a beginning, there must be at least one source node, and the existence of the end of a DAG implies there must be at least one sink node. **(T)**FINISH THIS **(end T)**

Lemma 2.1. *A finite DAG always has at least one source node and one sink node.*

Proof. Pick any node v_0 as a starting point. Now pick any one of its child nodes, say v_1 so (v_0, v_1) is an edge and a walk of network length one. We can now repeat this process adding some node v_2 , a child node of v_1 , to our walk to create a path of length two, (v_0, v_1, v_2) . We continue this process, producing a longer and longer walk, the sequence $(v_0, v_1, v_2, \dots, v_\ell)$. However at some point this has to stop and we are unable to grow the walk any further. First, because we have a DAG, there are no cycles so we can never return to a node already in our walk⁹. So at the very least, we will have placed every node in our walk. At this point, this last node cannot have any child nodes. If it did, this edge would connect to another node in our walk and that would define a cycle. Since we have a DAG that cannot be so. Of course,

⁸the term source node is also used to indicate the node at the start of an edge as shown in Fig. 1.4.

⁹So all walks on a DAG are always paths. We will note this more formally in section 2.4.

it may also be that the process stopped before every node was visited. In both cases, the final node v_ℓ in our sequence, when the process of lengthening the walk ends, must have no child nodes. That is this node v_ℓ is always sink node with $k^{(\text{out})}(v_\ell) = 0$.

We can repeat this argument but following the parents of every node, creating a walk $(v_i)_{i=-\ell}^0 = (v_{-\ell}, \dots, v_{-2}, v_{-1}, v_0)$. Now we are adding nodes to the start of the walk and again the process must eventually stop at which point the first node in the sequence $v_{-\ell}$ can have no parents, so case the node is a source with $k^{(\text{in})}(v_\ell) = 0$. \square

A directed graph with cycles can also have sources and sinks but it equally they are not required.

2.3 Useful DAG Subgraphs

First it is useful to note that any subgraph of a DAG is also a DAG.^{(T)¹⁰} This follows since any subgraph retains the direction of the edges so it is always a directed graph. Secondly as we only remove edges and nodes we cannot create any new cycles so the acyclic property remains.^{¹¹}

We define the PAST $\mathcal{D}^-(t)$ of a node $t \in \mathcal{V}$ to be the subgraph $\mathcal{D}^-(t) = (\mathcal{V}^-(t), \mathcal{E}^-(t))$ containing all nodes v (including t) which have a path which starts at v and ends at t (including the trivial path (t)) along with all the edges linking these nodes

$$\mathcal{V}^-(t) = \{v | v \preceq t\}, \quad \mathcal{E}^-(t) = \{(u, v) | (u, v) \in \mathcal{E}, u, v \in \mathcal{V}^-(t)\}. \quad (2.3.1)$$

The node t is therefore the unique sink node (the label t is for ‘target’) for the subgraph $\mathcal{D}^-(t)$. See Fig. 2.4 for an example.

Sometimes we just want the nodes in the past set and the term ‘past’ in the literature may refer to the nodes not the subgraph. The nodes in the past $\mathcal{V}^-(t)$ excluding the root node t are known as ANCESTOR nodes and we will denote the set of ancestor nodes of a root node t as $\tilde{\mathcal{V}}^-(t) \equiv \tilde{\mathcal{V}}_t^-$ where.

$$\tilde{\mathcal{V}}^-(t) = \{v | v \prec t\} = \mathcal{V}^-(t) \setminus \{t\}. \quad (2.3.2)$$

Likewise the FUTURE $\mathcal{D}^+(s)$ of a node s in the subgraph $\mathcal{D}^+(s) = (\mathcal{V}^+(s), \mathcal{E}^+(s))$ containing all nodes v which are on a path starting from node s and all the edges linking these nodes,

$$\mathcal{V}^+(s) = \{v | s \preceq v\}, \quad \mathcal{E}^+(s) = \{(u, v) | (u, v) \in \mathcal{E}, u, v \in \mathcal{V}^+(s)\}. \quad (2.3.3)$$

Note that the node s is the unique source node for this DAG $\mathcal{D}^+(s)$. The node s is therefore the unique source node for the subgraph $\mathcal{D}^+(s)$. Again Fig. 2.4 provides an example. Again if we want just the nodes in the future set of some node s other than the source node s , these are known as the descendants of so the set $\mathcal{V}^+(s) \setminus \{s\} = \{u | s \prec u\}$, are sometimes known as DESCENDANT nodes.

$$\tilde{\mathcal{V}}^+(s) = \{v | s \prec v\} = \mathcal{V}^+(s) \setminus \{s\}. \quad (2.3.4)$$

An INTERVAL $\mathcal{I}(s, t)$ is the subgraph defined by the intersection of the future of s and the past of t , i.e. $\mathcal{I}(s, t) = (\mathcal{V}^I(s, t), \mathcal{E}^I(s, t))$ where

$$\begin{aligned} \mathcal{V}^I(s, t) &= \{v | t \succeq v \succeq s\} = \mathcal{V}^+(s) \cap \mathcal{V}^-(t), \\ \mathcal{E}^I(s, t) &= \{(u, v) | (u, v) \in \mathcal{E}, u, v \in \mathcal{V}^I(s, t)\} = \mathcal{E}^+(s) \cap \mathcal{E}^-(t). \end{aligned} \quad (2.3.5)$$

That is $\mathcal{I}(s, t)$ is the set of all points and edges that lie on any path from the s to t . The interval is a proper subgraph and is also a DAG with a single source node s and a single sink node t . See Fig. 2.5 for an example.

¹⁰(T)Really? Do we need this here?

¹¹This second property indicates that we can start from a directed graph with cycles and by removing certain edges we can destroy all the cycles. Creating exact DAGs from directed graphs which have few cycles is not a unique process but is one of great practical interest.

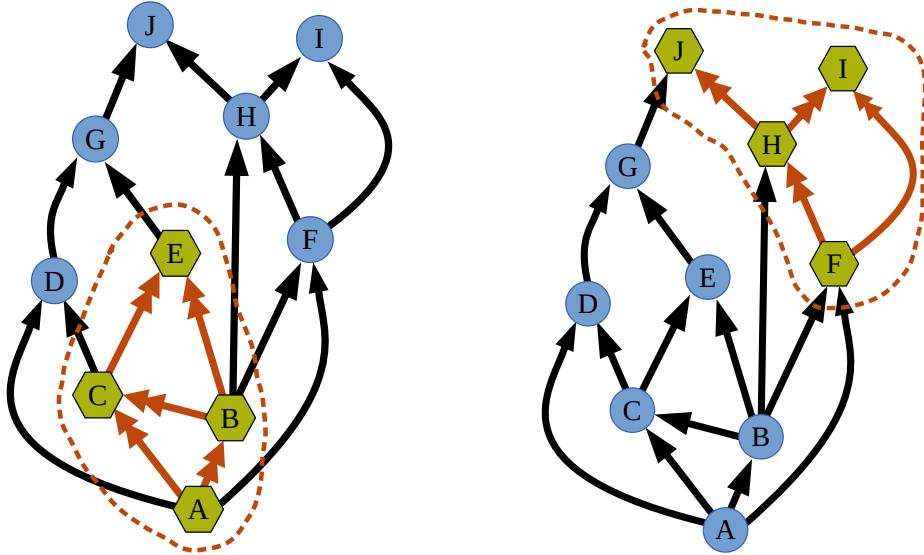


Figure 2.4: An illustration of the past and future DAGs. On the left we show the past of node E . This contains the four nodes shown as yellow hexagons, including node E , forming $\mathcal{V}^-(5) = \{A, B, C, E\}$, and all the edges $\mathcal{E}^-(E)$ between these nodes, shown as red double headed arrows. Together, these give the past DAG $\mathcal{D}^-(E) = (\mathcal{V}^-(E), \mathcal{E}^-(E))$ which is a sub-DAG for the larger DAG. In this case the past of node E is also an interval $\mathcal{I}[A, E]$. In the same way, on the right we show the future of node F which are the four nodes $\mathcal{V}^+(6) = \{F, H, I, J\}$ shown as yellow hexagons along with all the edges $\mathcal{E}^+(F)$ between these nodes, shown as red double headed arrows. Again the future sets are always sub-DAGs as shown here. In this case the future DAG of node F is not an interval as it has two sinks, nodes I and J .

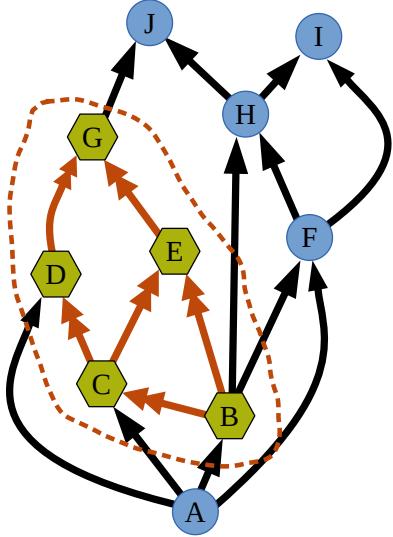


Figure 2.5: An illustration of an interval. Here the yellow hexagon nodes $\{B, C, D, E, G\}$ and all the red double headed arrows between them form the interval $\mathcal{I}(B, G)$. As with all intervals, this is a subgraph of the full network, has one source (B) and one sink (G) and is also a DAG. Note this subgraph is neither the past of node G (node A and three edges are missing) nor the future of node B (nodes F, H, I and J along with the relevant six edges are missing).

(T) Do we use open interval? If not delete this (end T)

Note it is sometimes useful to consider the OPEN INTERVAL, the set of points $\tilde{\mathcal{V}}^I(s, t)$, in which the unique source and sink of the interval, the defining nodes s and t , are excluded from the set of nodes:

$$\tilde{\mathcal{V}}^I(s, t) = \{v | t \succ v \succ s\} = \mathcal{V}^I(s, t) \setminus \{s, t\}. \quad (2.3.6)$$

2.4 Paths

The whole point of a network is to look beyond the nearest neighbours of each node. So let us look at the wider structure of a DAG and define some useful concepts.

We will use the notation $u \preceq v$ (**T**)¹² (**T**)¹³ to denote that there is a path from node u to a node v , i.e. that v is reachable from u . This includes the case where we have the trivial path $\mathcal{P} = (u)$ starting and ending at the same node so $u = v$. It is also convenient to introduce the notation \prec which excludes equality, so that $u \prec v$ indicates that there is a path from node u to a *distinct* node v , so $u \neq v$. For example, in Fig. 2.6 we have no edge (A, H) but there is a path from node A to node H so we can write $A \prec H$ and $A \preceq H$.

The reverse notation can be used in a consistent manner, so $v \succ u$ indicates there is a path ending at v coming from u with $u \neq v$. For our previous example we could also write that $H \succ A$ in Fig. 2.6.

A good starting point for our discussion is that walks are always paths on a DAG. This is simply because there are no cycles in a DAG. We defined walks, cycles and paths on a network in section 1.2.2. So for DAGs we will just talk of paths. Formally,

Lemma 2.2. *A walk on a DAG is always a path.*

Proof. If we had a walk where two nodes appeared twice in the sequence of nodes then that part of the sequence defines a cycle. That is if $\mathcal{W} = (v_i)_{i=0}^{\ell}$ where we have $v_j = v_k$ for some $0 \leq j < k \leq \ell$, then the subsequence $\mathcal{C} = (v_i)_{i=j}^k$ is a cycle. Since a DAG has no cycles, we can never have this situation, so no nodes can ever be repeated on any walk in a DAG, i.e. walks are always paths. \square

Sometimes it is important to indicate when there is no path between two nodes. Most of the time we do this by absence, we just don't write anything if there is no path between node u and v . If we want to be more explicit then we use the negated symbol $\not\prec$ as in $u \not\prec v$ shows there is no path from u to v . For instance in Fig. 2.6 we could state explicitly that $D \not\prec F$ or indeed that $H \not\prec A$ (the paths are from A to H and not the other way round as the notation here requires).

In some situations, we will want to include the trivial path of one node, the path from a node to itself, as part of some argument. To do this we will use the notation $u \preceq v$ to indicate that there is a path from u to some node v including the trivial path where $v = u$. So in Fig. 2.6 we have that $H \preceq H$ and so forth.

2.4.1 The Longest Path

(T) Maybe leave distance till another section. (**end T**)

The longest path between two nodes is the path with the largest number of edges, the path with the largest length ℓ , the largest number of edges in the path.

One of the most interesting ideas in a DAG is the realisation that the LONGEST PATH between two nodes in the DAG is a useful and well defined concept, for example see Fig. 2.6. Again, length here is the network length defined to be the number of edges in a path. In most ordinary networks, such as a social network, you can find a path between two nodes that visits most other nodes of degree two or more in a network. So the length of the longest path in an ordinary network between two nodes is likely close to the number of nodes N in the network. For instance if we ignored the edge direction in the DAG example in Fig. 2.6 we can find a path that visits every node such as $(A, B, F, I, H, J, G, D, C, E)$.

¹²(T) We only say “ u precedes v ” when there is an edge ?

¹³(T) In some real sense, given the arrow of time in a DAG, v is later than u .

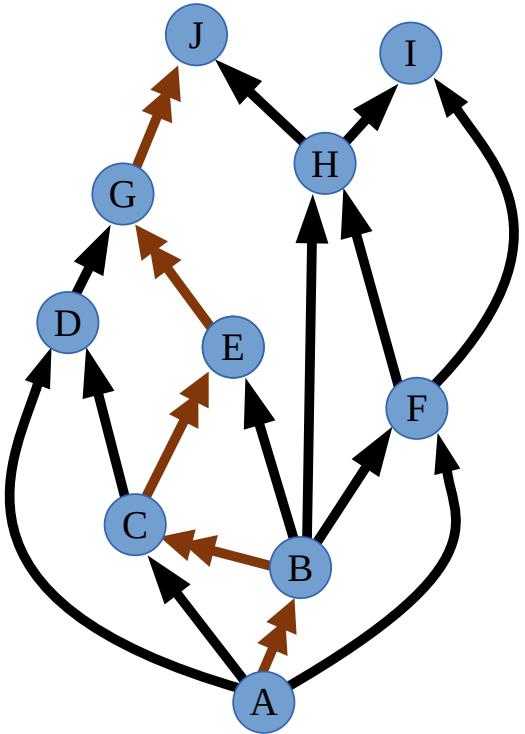


Figure 2.6: An illustration of the longest paths in a DAG using the same DAG as Fig. 2.3. The double headed red arrows show an example of one of the longest paths in this DAG, namely (A, B, C, E, G, J) , of length 5. However, this is not unique as we can substitute node D for node E and we have a path of the same length. By way of contrast, the shortest paths from A to J have length 3, for example (A, F, H, J) but there are others of this same length.

However, in a DAG the longest path is not unhelpfully long making it a useful alternative in a DAG to the *shortest path*. In particular, in a DAG we can usefully define lengths and distances from the length of longest paths, providing an alternative to the more traditional shortest-path length definition.

To see this, imagine we have a source node u and a target node v in mind for a path in a DAG and we want the longest path between them. Suppose we have a path starting at u and we keep growing this path to try to reach v but taking in as many other nodes as possible. The order in the network means we can never go ‘backwards’. That is every time we add another node to the end of a path, by definition that additional node has to be a node that is ‘later’ than all other nodes in the existing path. We can now never visit nodes at earlier positions in the order of nodes in the DAG. At some point you will either reach the target node or miss it and we have to start again. The order in a DAG imposes such a strong constraint that usually there is a very limited set of nodes we can possibly visit in a path and each time we add another node w to our path growing out from the source node u , we exclude some more nodes which are earlier than w . Contrast this behaviour in a DAG with a social network where there is no sense of order. In a simple network, we can add nodes to a path growing out from a source node u which are closer or further way from the target node v , e.g. as measured by shortest path distance to v , but we can often find paths that move closer to the target node v then move away again, perhaps many times.

You can still measure path lengths and the distance between nodes in a DAG using shortest path lengths. However, the shortest paths do not seem to probe the constraint of the node order in a DAG. It is the longest path that seems to do that and as such, this seems to be the more important measure of length in a DAG. (T)¹⁴ We can see this explicitly in some of the simple models we consider later. However, for most DAGs the conjecture that longest-path lengths are more important than shortest paths lengths remains just that, a supposition that needs to be considered for each case. The length of various longest paths in a DAG does is the basis for some key definitions which give critical information used when analysing a DAG.

¹⁴(T) REALLY?

So we will define $d_{vu}^{(\text{lp})}$ to be the length of the longest path from node u to node v . This is only defined when there is a path from u to v , that is when $u \preceq v$. In particular in our definition only one of $d_{vu}^{(\text{lp})}$ and $d_{uv}^{(\text{lp})}$ can be defined if u and v are distinct. For this reason using the longest path length to give the “distance” between two nodes in a DAG is *not* a distance in the strict mathematical sense as it breaks non-negativity (3.3.1a) and symmetry (3.3.1b). For brevity, we may sometimes refer to $d_{vu}^{(\text{lp})}$ as the “distance” between two connected nodes u and v . However, because it breaks some of the properties of a distance at least some of the time, we should be aware this measure $d_{vu}^{(\text{lp})}$ is what is strictly called a PSEUDO-DISTANCE.

2.5 Height and Depth

The height and depth are properties of every node in a DAG that reflects the global position of that node in the DAG in terms of the inherent order of the DAG. Essentially the height measures the distance of the node from the start of the DAG and the depth is the distance from the end of the DAG. They do not have any equivalent in the analysis of standard networks, only DAGs have a beginning and an end which is inevitable given their order. Height and depth are quick and easy to calculate despite the fact they are node measures that depend on the structure of the whole DAG, not just the region close to the node.

Definition 2.5.1— Height and Depth

The HEIGHT $h(v)$ of a node v in a DAG \mathcal{D} is the largest length of any path $\mathcal{P}(u, v)$ to that node v from any other node u

$$h(v) = \max(|\mathcal{P}(u, v)| - 1 \mid u \preceq v). \quad (2.5.1)$$

The DEPTH $d(u)$ of a node u in a DAG \mathcal{D} and this is the length of the longest path from that node u to any of the sink nodes $t \in \mathcal{T}$, that is

$$d(u) = \max(|\mathcal{P}(u, v)| - 1 \mid u \preceq v) \quad (2.5.2)$$

Let us focus first on height. In fact the height is always defined in terms of the length of a longest path from a source node. First we need only consider the longest paths between u and v as we are finding the maximum possible path length. Then if any path starting from a node u that was not a source, that path can always be extended by one more node as there must be an incoming edge to u and so a predecessor to u . Repeating this argument we find the longest paths to v that define height must always start from a source node. This shows that the height is always defined as we can always follow incoming edges back to a source. Even if there are no incoming edges to a node, i.e. it is a source node, as then the only path to that node is from itself to itself. This is the trivial path of length zero, so the height of a source node is always zero. So we can also write this as

$$h(v) = \max(d_{sv}^{(\text{lp})} \mid s \in \mathcal{S}). \quad (2.5.3)$$

Examples of heights are shown in Fig. 2.7.

The definition of height tells us that the height of one node u is always smaller than any distinct node v if there is a path from u to v , so $h(u) < h(v)$ if $u \prec v$. This is because any path from u to v is at least length one (since u and v are distinct) and we know there is at least one path from some source node to u of length equal to $h(u)$. By combining these two paths we have a path from a source node to v of length no less than $h(u) + 1$, so $h(v) > h(u)$.

We can go a step further and if the longest path from u to v is of length $d_{vu}^{(\text{lp})}$ then there is a path from a source to v via u of length $h(u) + d_{vu}^{(\text{lp})}$ so that $h(v) \geq h(u) + d_{vu}^{(\text{lp})}$. The $(h(u) + d_{vu}^{(\text{lp})})$ is only a lower bound since there may be paths from sources to v which do not go via u and which are longer than any path from a source via u to v . For example see nodes H and J in Fig. 2.7 where their heights are defined from the single source node A . So we only know that

$$h(v) \geq h(u) + d_{vu}^{(\text{lp})} > h(u) \quad \text{if } u \prec v. \quad (2.5.4)$$

Put another way, the network length of the longest path between the source and target nodes of an edge need not equal the difference of heights at the two ends of the edge. More generally, for any two nodes where $v \succeq u$

$$d^{(\text{lp})}(v, u) \leq h(v) - h(u) \quad \text{where } v \succeq u. \quad (2.5.5)$$

Now let us consider the complementary measures, the DEPTH $d(u)$ (2.5.2) of a node u in a DAG \mathcal{D} and this is the length of the longest path *from* that node u to any of the sink nodes $t \in \mathcal{T}$, that is

$$d(u) = \max(|\mathcal{P}(u, v)| - 1 \mid u \preceq v) = \max(d_{tu}^{(\text{lp})} \mid t \in \mathcal{T}). \quad (2.5.6)$$

Again the algebraic definition is written in term of all nodes v but, as stated in words, only the sink nodes will give the longest paths definition depth. The depth of each node in Fig. 2.7 is also shown. Using the same arguments as we did for height, we deduce that

$$d(u) \geq d(v) + d_{vu}^{(\text{lp})} > d(v) \quad \text{if } u \prec v. \quad (2.5.7)$$

We can extend these ideas from nodes to the whole DAG and we define the height $h(\mathcal{D})$ of a DAG which to be the largest height of any node in the DAG. This height comes from the length of a path from a source node, as noted above in the definition of the height of any node. However, if this path ends on a node v that is not a sink node, we can always extend the path to a successor node of v , in which case the path defining the height of a DAG cannot be to such a node v . This contradiction shows that the height is always the length of a longest path from a source node to a sink node. Not all source and sink node pairs will have such a longest path between them, but at least one pair will. We can see an example of a path which defines the height of a DAG in Fig. 2.6.

We can see that the largest depth of a DAG, say $d(\mathcal{D})$, is given by the length of the same path that defines the height H of the DAG, so $H = h(\mathcal{D}) = d(\mathcal{D})$. Again we can see this result in the example in Fig. 2.6.

We can use the network length of longest paths, the height and depth of each node, in various ways when analysing a DAG and we will return to these ideas in section ???. We can also use the height and depth of a node as coordinate, a convenient way to start visualising a DAG, see Fig. 6.5 in section 6.5.

2.6 Total Order

(T)Let us use ‘total order’ in these notes as it makes more sense than total order. We will use ‘topological sort’ for the algorithm. **(end T)** **(T)**Terminology? Topological sort seems to be the language of computer science, total order etc from maths and posets **(end T)**

A TOTAL ORDER for a DAG is a sequence of nodes¹⁵ $\mathcal{T} = (v_t)_{t=1}^N$ built from all the nodes in the DAG and where any edge in the DAG is from a node earlier in the sequence to one that

¹⁵In practice the index t of the sequence is often related to the order coming from time so we use the letter t as an index here. These are not in general target nodes such as sinks whose index we often wrote as the same letter t .

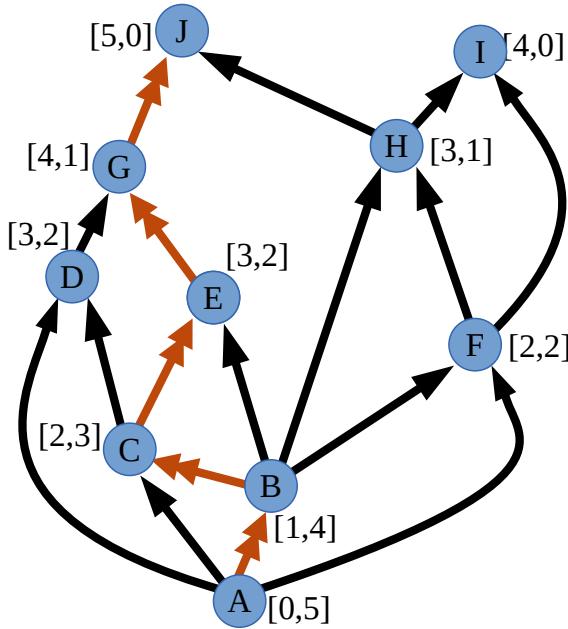


Figure 2.7: An illustration of height, depth and criticality of nodes in the same DAG as used in Fig. 2.3 and Fig. 2.6. Beside each node v the height and depth of that node are given as $[h(v), d(v)]$. The double headed red arrows show an example of one of the longest paths in this DAG, namely (A, B, C, E, G, J) , of length 5. However, this is not unique as we can substitute node D for node E and we have a path of the same length. The height of the DAG is the length of the longest path between any two nodes, always between a source and a sink. Our example path is an example of a longest path that defines the height H of the DAG, here $H = 5$. Note that the way we have laid out the nodes also illustrates how we can use criticality and height to set the rough coordinates of nodes to arrive at a reasonable visualisation of a DAG.

is later. That is if $(v_r, v_t) \in \mathcal{E}$ is an edge then $r < t$. Conversely, we can *only* have an edge from v_r to v_t if $r < t$ but this is not guaranteed to exist. Taking this further, we are not even guaranteed there is any path from node v_t to node v_r if $r < t$. So the total order respects the directions in the DAG but does not define all the edges.

A total order on a DAG is equivalent to what is also called a total order on a poset (there also known as a ‘linear order’). Here the poset is the one related to every DAG but we will postpone a discussion of those concepts until section 2.9. In computer science an algorithm used to find a total order on a DAG is called a topological sort so a total order on a DAG is also known as a ‘topological order’.

One of the key properties of a DAG is that there is always at least one total order for any DAG.

Lemma 2.3. *Every DAG has at least one total order.*

Proof. The easiest way to prove this is to give an iterative algorithm to construct a total order for any DAG $\mathcal{D} = (\mathcal{V}, \mathcal{E})$.

Algorithm 2.6.i. Topological Sort

Suppose we count the steps in our procedure with the integer index τ . At any step τ in our algorithm we have a sequence of nodes \mathcal{T}_τ and a DAG $\mathcal{D}_\tau = (\mathcal{V}_\tau, \mathcal{E}_\tau)$ where each node in the original DAG \mathcal{D} is either in the sequence \mathcal{T}_τ or it is in the node set of the current DAG \mathcal{D}_τ , so $\mathcal{T}_\tau \cap \mathcal{V}_\tau = \emptyset$ and $\mathcal{T}_\tau \cup \mathcal{V}_\tau = \mathcal{V}$. The current edge set \mathcal{E}_τ is the set of all edges which are both in the original DAG and which run between nodes in the current node set \mathcal{V}_τ , so $\mathcal{E}_\tau = \{(u, v) | u, v \in \mathcal{V}', (u, v) \in \mathcal{E}\}$.

1. Start with $\tau = 0$. Our first sequence is an empty sequence $\mathcal{T}_{\tau=0}$ and the initial DAG \mathcal{D}_τ is a copy of the full DAG \mathcal{D} .
2. Start a new step by incrementing our counter τ by one.
3. Search for all nodes in the current DAG \mathcal{D}_τ which have no-incoming edges, source nodes \mathcal{S}_τ in \mathcal{D}_τ .
4. If \mathcal{S}_τ is empty then we have a cycle present and we have to stop this algorithm. This
5. We use these source nodes \mathcal{S}_τ to extend the current sequence \mathcal{T}_τ to give us the sequence $\mathcal{T}_{\tau+1}$ for the next step.
6. Remove these source nodes \mathcal{S}_τ from the current node set \mathcal{V}_τ creating the node set for the next step $\mathcal{V}_{\tau+1}$. So $\mathcal{V}_{\tau+1} = \mathcal{V}_\tau \setminus \mathcal{S}_\tau$.
7. The next edge set $\mathcal{E}_{\tau+1}$ is obtained by deleting all the edges in \mathcal{E}_τ which start from any of the source nodes \mathcal{S}_τ . That is equivalent to $\mathcal{E}_{\tau+1} = \{(u, v) | (u, v) \in \mathcal{E}, u, v \in \mathcal{V}_{\tau+1}\}$.
8. If the next node set $\mathcal{V}_{\tau+1}$ is not empty then we repeat the process from step 2.
9. The last sequence we construct, which is $\mathcal{T}_{\tau+1}$ at this point, is the total order for the original DAG \mathcal{D} .

topological sort algorithm to find a total order is illustrated in Fig. 2.8. \square

This algorithm illustrates a few other points. First, if at any stage, our current DAG \mathcal{D} has nodes $|\mathcal{V}_\tau| > 0$ but we can find no source nodes, $\mathcal{S}_\tau = \emptyset$, then we must have a cycle and we cannot have a DAG. This is flagged in step 4. So this algorithm is also a test to see if a network has cycles.

Also note that whenever we have more than one node to add from the set \mathcal{S}_τ in step 5 we can extend the sequence \mathcal{T}_τ with these nodes in any order. As a result this algorithm can produce many different total orders. So in general, a DAG does not have a unique total order. More precisely this algorithm can find $\prod_\tau |\mathcal{S}_\tau|$ distinct total orders. However, this is still not usually all possible total orders. The example in Fig. 2.8 has one total order for the DAG, the nodes in alphabetical order, that cannot be found by this algorithm.

If the height of each node is known then we can construct a total from those values. Specifically, we can create a sequence of nodes $(v_t)_{t=1}^N$ such that the height never decreases as we go along the sequence, so $h(v_r) \leq h(v_t)$ for all $r < t$. This sequence is a total order¹⁶. This follows from the property stated in (2.5.4) that the difference in height between the source u and target v node of an edge $(u, v) \in \mathcal{E}$ always increases along an edge, so $h(u) < h(v)$. This is also applied to pair of nodes connected by a non-trivial path so $u \prec v \Rightarrow h(u) < h(v)$.

However, nodes of the same height can appear in any order relative to each other in any total order since there can be no edge between such nodes. For the example of a total order

¹⁶As usual, we could also use the depth of each node but then depth of nodes in the sequence must not increase, $d(v_r) \geq d(v_t)$ for all $r < t$. This will not always give the same total orders as the height based total order.

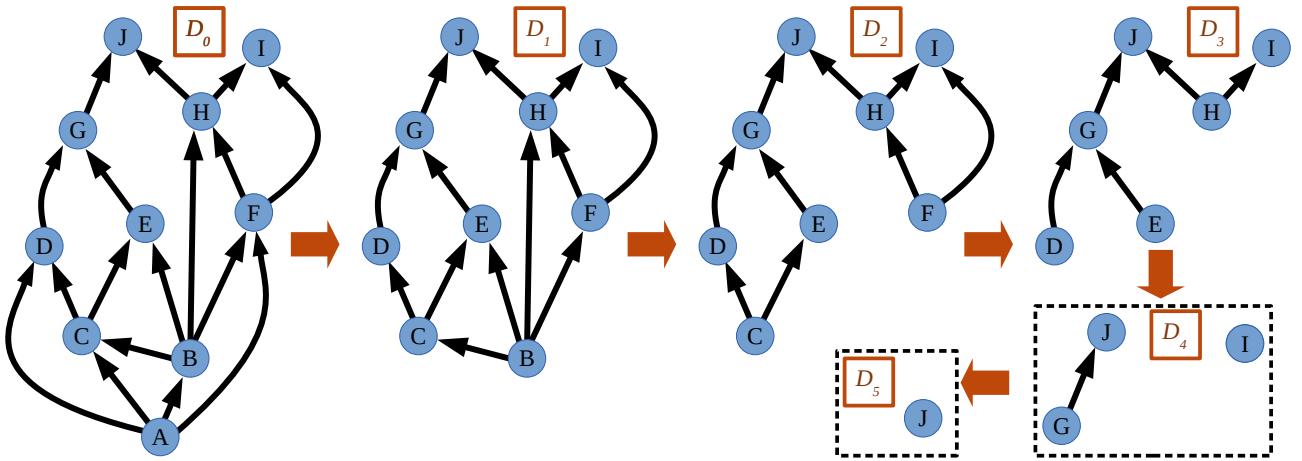


Figure 2.8: An example of the sequence of DAGs \mathcal{D}_τ produced by the topological sort algorithm 2.6.i in section 2.6 used to produce a total order for the DAG. The set of nodes eliminated when moving from $\mathcal{D}_{\tau-1}$ to \mathcal{D}_τ are \mathcal{S}_τ where $\mathcal{S}_1 = \{A\}$, $\mathcal{S}_2 = \{B\}$, $\mathcal{S}_3 = \{C, F\}$, $\mathcal{S}_4 = \{D, E, H\}$, $\mathcal{S}_5 = \{G, I\}$ and $\mathcal{S}_6 = \{J\}$. This gives us one possible total order for the DAG as the sequence $(A, B, C, F, D, E, H, G, I, J)$. However note we can place the nodes from each set \mathcal{S}_τ in any order in the total order so even within this algorithm we can produce $2 \times 3 \times 2 = 12$ different total orders. Even then, this is not all the possible total orders. The sequence with the nodes in alphabetical order, $(A, B, C, D, E, F, G, H, I, J)$ is also a total order but this algorithm will never find it. For instance node F from \mathcal{S}_3 is always placed in the total order before nodes D and E which are in \mathcal{S}_4 .

built in height order, such total orders can be created in $\prod_{h=0}^H |\mathcal{V}_h|$ different ways where \mathcal{V}_h is the subset of nodes of height h . More generally, any two nodes not connected by a path can have any height values relative to each other. Without some path between them there is no constraint on their relative position in any total order whatever their height values are. So using height to define a total order is over constrained and there can be total orders where nodes are node in height order.

For instance, in Fig. 2.9 we have that the subsets of the nodes by height¹⁷ $\mathcal{V}_0 = \{A\}$, $\mathcal{V}_1 = \{B\}$, $\mathcal{V}_2 = \{C, D\}$, $\mathcal{V}_3 = \{E, F, H\}$, $\mathcal{V}_4 = \{G, I\}$ and $\mathcal{V}_5 = \{J\}$ where \mathcal{V}_h is the set of nodes of height h . There are $1 \times 1 \times 2 \times 3 \times 2 \times 1 = 12$ ways to create the total orders by height with one example being $\mathcal{T} = (A, B, C, D, E, F, H, G, I, J)$. Now we see a clear mismatch between the partition of the nodes into the \mathcal{S}_τ found by the Algorithm 2.6.i and a total order built from these height subsets \mathcal{V}_h . In particular, node $D \in \mathcal{S}_4$ is always later than $F \in \mathcal{S}_3$ in the total orders found using the total order algorithm above, Algorithm 2.6.i, but D (height 2) is always before F (height 3) in any total order formed from the height node sets. Again note the total order with nodes in alphabetical order is not generated by this height total order either as the latter always puts node H of height 3 before node G of height 4.

Finally we note that the only DAGs which are weakly connected and which have a unique total order are those where there is a subgraph which can be visualised as a single line, a path, running through all the nodes. This path is then the unique total order.

To show this we can start with the observation that if we have a total order $\mathcal{T} = (v_t)_{t=1}^N$ then if there is a path from node v_r to node v_s then we must have that $r < s$. More formally

Lemma 2.4. *For any given a total order $\mathcal{T} = (v_t)_{t=1}^N$*

$$v_r \prec v_s \quad \Rightarrow \quad r < s. \quad (2.6.1)$$

To show this suppose we have that there is a path \mathcal{P} from node v_r to a distinct node v_s (denoted $v_r \prec v_s$) where $r < s$. If we have an edge from node v_s to a distinct node v_t we know

¹⁷An example of a partition of the set of nodes.

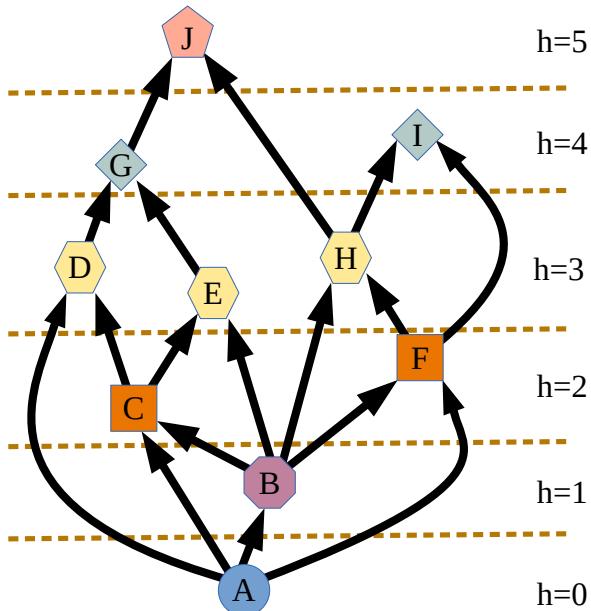


Figure 2.9: An illustration of nodes arranged by height in the same DAG as used in Fig. 2.3 and Fig. 2.8. On the right is the height h of nodes within each region bounded by the dashed lines. Nodes of the same height are shown using the same node colour and shape. A total order \mathcal{T} can be constructed from the height if the sequence of nodes in the total order \mathcal{T} is in order of height. So all of the height two nodes $V_2 = \{C, F\}$ to \mathcal{T} appear before the height three nodes $V_3 = \{D, E, H\}$ in the sequence \mathcal{T} . With nodes of the same height, these can appear in any order relative to each other. For instance here the third and fourth nodes in the total order are C and F but in either order. Again the total order defined by putting the nodes in alphabetical order is not one of the total orders created by height. For instance, node G of height 4 is always after node H of height 3 in these height total orders. Note how height has been used to lay out the nodes to give a reasonable visualisation of a DAG.

by definition of the total order that $s < t$. However we can add node v_t to the end of the path \mathcal{P} from v_r to v_s to produce another legal path \mathcal{P}' from node v_r to a distinct node v_t (denoted $v_r \prec v_t$) where we know that $r < t$ and never $r \geq t$. So if the lemma is true for any given path, it is true for all possible extensions of the path by one more node. From the definition of a total order we know that the lemma is true when there is an edge (r, s) . So we know we can extend this edge to find paths of all possible lengths starting at r and the lemma is true. Since the starting point r was arbitrary, we have shown the lemma to hold for all possible paths.

Note that we can also see that there can be no path from node v_r to a distinct node v_s when $r > s$. That is

Lemma 2.5. For any given a total order $\mathcal{T} = (v_t)_{t=1}^N$

$$r > s \quad \Rightarrow \quad v_r \not\prec v_s . \quad (2.6.2)$$

We can look at the previous proof and see that the initial edge in any path is always (v_r, v_s) with $r < s$, never $r \geq t$, because of the definition of a total order. The case $r = s$ is the trivial case. For all other cases, $r > s$ and then every edge we add to ??? can only increase the index for the same reason. So there can never be a path from a node v_r to a node v_s if node v_r is later in any total order than v_s .

2.7 Transitive Reduction and Transitive Completion.

(T) What properties are invariant under TC TR? Poset properties. (end T)

The connectivity of a DAG finally leads us to describe properties and transformations on the whole network. A DAG is TRANSITIVELY COMPLETE if there is an edge between any pair of nodes where there is a path or connection between the nodes. Obviously the edge has to be in the same direction as the path. That is if $u \prec v$ then $(u, v) \in \mathcal{E}$. This definition seems a little redundant without a context since by definition every edge every edge is a path in its own

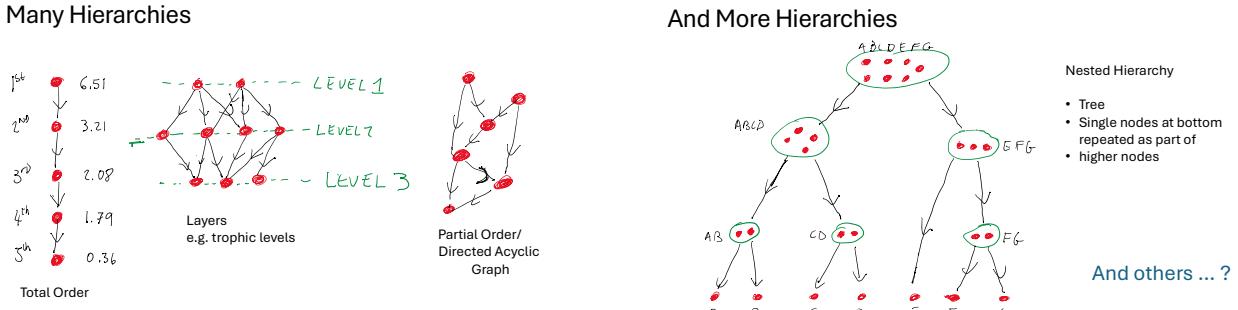


Figure 2.10: An illustration of some various hierarchies realised as different types of directed networks. In the three networks at the top are all DAGs. Each node usually represents a distinct object that which appears in only one node. The lower network is an example of a nested hierarchy network. This is a tree so is a special type of network but now each node in the tree is a set of the source nodes in the past of the tree node. So this is a special case of a set inclusion DAG.

right so $(u, v) \in \mathcal{E} \Rightarrow u \prec v$. The main use is in the ability to create and discuss a transitively complete DAG $\mathcal{D}^{(tc)}$ associated with any DAG \mathcal{D} . That is for any given DAG \mathcal{D} we add edges so that if there is a path from u to v then in the transitively complete graph $\mathcal{D}^{(tc)}$ there is always an edge from u to v

$$\mathcal{D} = (\mathcal{V}, \mathcal{E}) \rightarrow \mathcal{D}^{(tc)} = (\mathcal{V}, \mathcal{E}^{(tc)}) , \quad \mathcal{E}^{(tc)} = \{(u, v) | u \prec v, u, v \in \mathcal{V}\} . \quad (2.7.1)$$

There is a unique transitively complete DAG $\mathcal{D}^{(tc)}$ for every DAG \mathcal{D} , but most transitively complete DAGs $\mathcal{D}^{(tc)}$ can be created from the transitive completion of many different DAGs \mathcal{D} . Put another way, there is a family of DAGs $\mathcal{F} = \{\mathcal{D}_i\}$, in which each member \mathcal{D}_i has the same transitively complete DAG $\mathcal{D}^{(tc)}$. Since we only add edges to reach a transitively complete DAG, the largest (most edges) DAG of the DAGs \mathcal{D}_i in one family $\mathcal{D}^{(F)}$ is the transitively complete DAG $\mathcal{D}^{(tc)}$ of all the DAGs in that family.

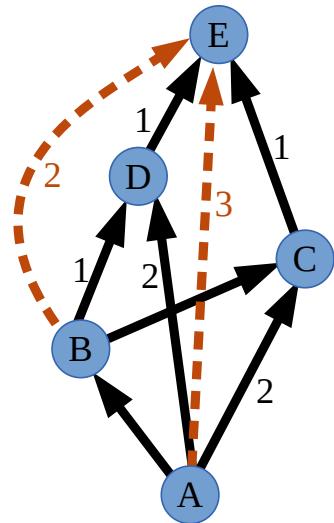


Figure 2.11: A transitively complete DAG. The edges shown as red dashed lines are part of this transitively complete DAG. These red dashed edges are highlighted as they are the ones which are added by transitive completion to the DAG shown in Fig. 2.1a. Note the edges added to complete a DAG always have a network length (see section 3.3.2) greater than 1. Here the extra edge on the left has network length 2 while the central edge added has network length 3.

If the transitively complete DAG is the largest of this family of DAGs, then there is also a minimum sized DAG in this set. This is found by making the TRANSITIVE REDUCTION of any of the DAGs in the family. The transitive reduction of a DAG, from \mathcal{D} to $\mathcal{D}^{(tr)}$, is where we remove any edge if there is a path of length two or more between the nodes connected by that

edge. Put another way, if we remove any edge in the transitively reduced DAG $\mathcal{D}^{(\text{tr})}$ we will destroy the connection between at least one pair of nodes. For instance, in Fig. 2.11, removing the edge from node A to node D does not change the node-node connectivity of the remaining graph. This means we can use the network length $\ell_{(u,v)}$ of an edge (u,v) defined in section 3.3.2 to give a formal definition of the transitively reduced DAG $\mathcal{D}^{(\text{tr})}$

$$\mathcal{D} = (\mathcal{V}, \mathcal{E}) \rightarrow \mathcal{D}^{(\text{tr})} = (\mathcal{V}, \mathcal{E}^{(\text{tr})}), \quad \mathcal{E}^{(\text{tr})} = \{(u,v) | \ell_{(u,v)} = 1, u, v \in \mathcal{V}\}. \quad (2.7.2)$$

The edges in the transitively reduced DAG are essential to ensure the connectivity in the reduced DAG $\mathcal{D}^{(\text{tr})}$ is the same as in the original DAG \mathcal{D} , so if $u \prec v$ in \mathcal{D} the same relation holds in $\mathcal{D}^{(\text{tr})}$. Again we find that the subset of edges needed to move from a DAG to its transitive reduction $\mathcal{D}^{(\text{tr})}$, is unique for every DAG. We call the edges of a DAG \mathcal{D} that cannot be removed without changing the connectivity, the edge set of the transitively reduced DAG $\mathcal{D}^{(\text{tr})}$, the ESSENTIAL EDGES of the DAG \mathcal{D} . So an edge such as (A, B) in Fig. 2.12 is an essential edge. Likewise, any edge not needed to preserve the connectivity, such as (A, D) in Fig. 2.12, is called an INESSENTIAL EDGE.

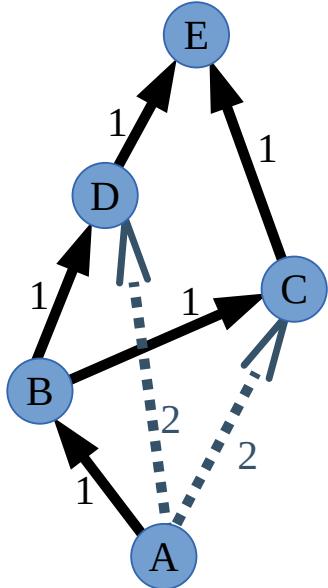


Figure 2.12: A transitively reduced DAG. The edges shown as blue dashed lines are not part of this transitively reduced DAG. These blue dashed edges are inessential edges of the DAG shown in Fig. 2.1a and these are removed in the process of transitive reduction. They are edges linking nodes separated by a longest path of network length 2 (see section 3.3.2). All the edges in the transitively reduced graph (solid black lines) connect nodes separated by a longest path of length 1 which has to be a path via the edge itself.

Again we can define a family of DAGs, $\mathcal{F} = \{\mathcal{D}_i\}$ where the transitive reduction of any DAG in this family is the same DAG $\mathcal{D}^{(\text{tr})}$, and that is the smallest (least number of edges) of the family. Finally, this is the same as the family of DAGs sharing one transitively complete DAG. That every DAG \mathcal{D} is part of a family of DAGs, $\mathcal{F} = \{\mathcal{D}_i\}$ and that family is the set of all DAGs which share the same transitively complete DAG $\mathcal{D}^{(\text{tc})}$ and the same transitively reduced DAG $\mathcal{D}^{(\text{tr})}$.

This family of DAGs is related to subsets of the sets of INESSENTIAL EDGES: those edges which are present in the transitively complete graph but not in the transitively reduced graph of a DAG family. For any subset of the inessential edges, we add those to the transitively reduced graph to find another member of this family of DAGs. That is this family is equivalent to the power set of the set of inessential edges, for example see Fig. 2.13. As power sets can be seen as a DAG, so our family of related DAGs is in some sense a DAG OF DAGS but this does not seem to be a helpful observation.

We can also use the idea of power sets and inessential edges with the focus on a DAG that has some but all the inessential edges present which is what we have from a typical data set. So suppose we have a DAG $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ and now we split the edge set \mathcal{E} into two parts, the essential edges $\mathcal{E}^{(\text{ess})}$ and inessential $\mathcal{E}^{(\text{iness})}$ in the DAG (so that $\mathcal{E}^{(\text{ess})} \cup \mathcal{E}^{(\text{iness})} = \mathcal{E}$ and

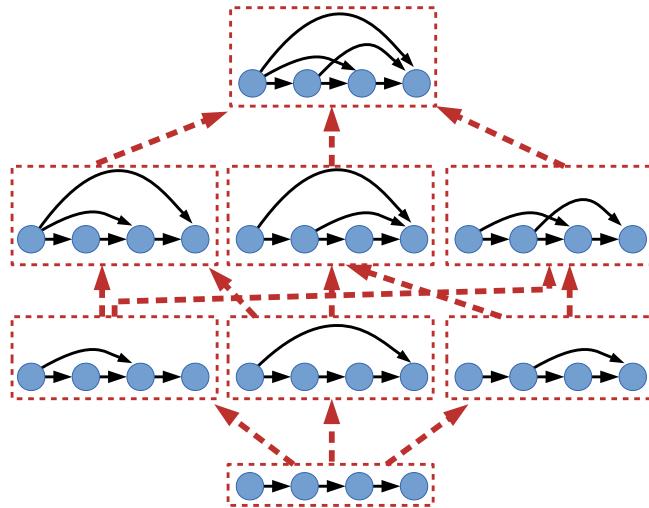


Figure 2.13: An illustration of a DAG of DAGs. Every DAG is a part of a family of DAGs in which each DAG is related to the same poset. Equally no other DAG is related to this one poset. In network terms, the transitively reduced DAG is the same for all the DAGs in the family. The only difference is which of the three possible inessential edges they contain. The inessential edges are the edges which are in the transitively complete graph at the top but not in the transitively complete graph at the bottom, i.e. these are the edges shown as curved lines. So each distinct DAG in the family contains a different subset of possible inessential edges. So this pattern can also be thought of as the power set of inessential edges. The smallest DAG at the bottom, the transitively reduced DAG, is the Hasse diagram of the poset. The largest, at the top, is the transitive completion of any of these DAGs, and in this DAG there is an edge for every relation in the poset. Note this example is a special case where the DAGs have a unique total order, the order clear in the bottom transitively reduced DAG.

$\mathcal{E}^{(\text{ess})} \cap \mathcal{E}^{(\text{iness})} = \emptyset$). If we just keep the essential edges we have the transitively reduced graph so $\mathcal{E}^{(\text{tr})} = \mathcal{E}^{(\text{iness})}$ and $\mathcal{D}^{(\text{tr})} = (\mathcal{V}, \mathcal{E}^{(\text{ess})})$. On the other hand we can add any number of the inessential edges in \mathcal{D} and still get a DAG within the same family of DAGs sharing the same poset structure. Equally we have a similar mini-family of DAGs by adding the edges $\mathcal{E}^{(\text{rest})}$ that are in the transitively complete graph $\mathcal{D}^{(\text{tc})} = (\mathcal{V}, \mathcal{E}^{(\text{tc})})$ but are not in the original DAG \mathcal{D} . These edges $\mathcal{E}^{(\text{rest})}$ are a subset of all the essential edges $\mathcal{E}^{(\text{tc})}$ in the transitively complete graph ($\mathcal{E}^{(\text{tc})} = \mathcal{E}^{(\text{rest})} \cup \mathcal{E}$, $\mathcal{E}^{(\text{rest})} \cap \mathcal{E} = \emptyset$).

One possible use is that these families of DAGs can represent similar DAGs that might capture some of the uncertainty in the data while all the time retaining the same core logical connectivity i.e. al all equivalent to the same poset.

What they do is emphasise that any one DAG is more than a poset and it carries more information than the corresponding poset. Each DAG occupies one point in a space of the family of DAGs (an equivalence class of DAGs).

Now we can talk about the power set of the inessential edges $\mathcal{E}^{(\text{iness})} \dots$

If an edge (u, v) is inessential, then by definition there must be an alternative path linking the two ends of the edge, from u to v . Such an alternative path must not be direct, it must have at least one other node on it so such an alternative path must be of length two or more. In this case our direct edge (u, v) would carry a weight of two or more. The converse is that if there is no alternative route from u to v the the edge (u, v) is essential and equally the distance between the nodes is one and this is the weight for that edge.

So in a transitive reduced DAG all the edges have length one and if there is any edge of length greater than one then the graph is not transitively complete.

The network length of an edge also defines a sequence of DAGs which we can construct from any DAG \mathcal{D} in which every DAG has same connectivity between any pair of nodes but

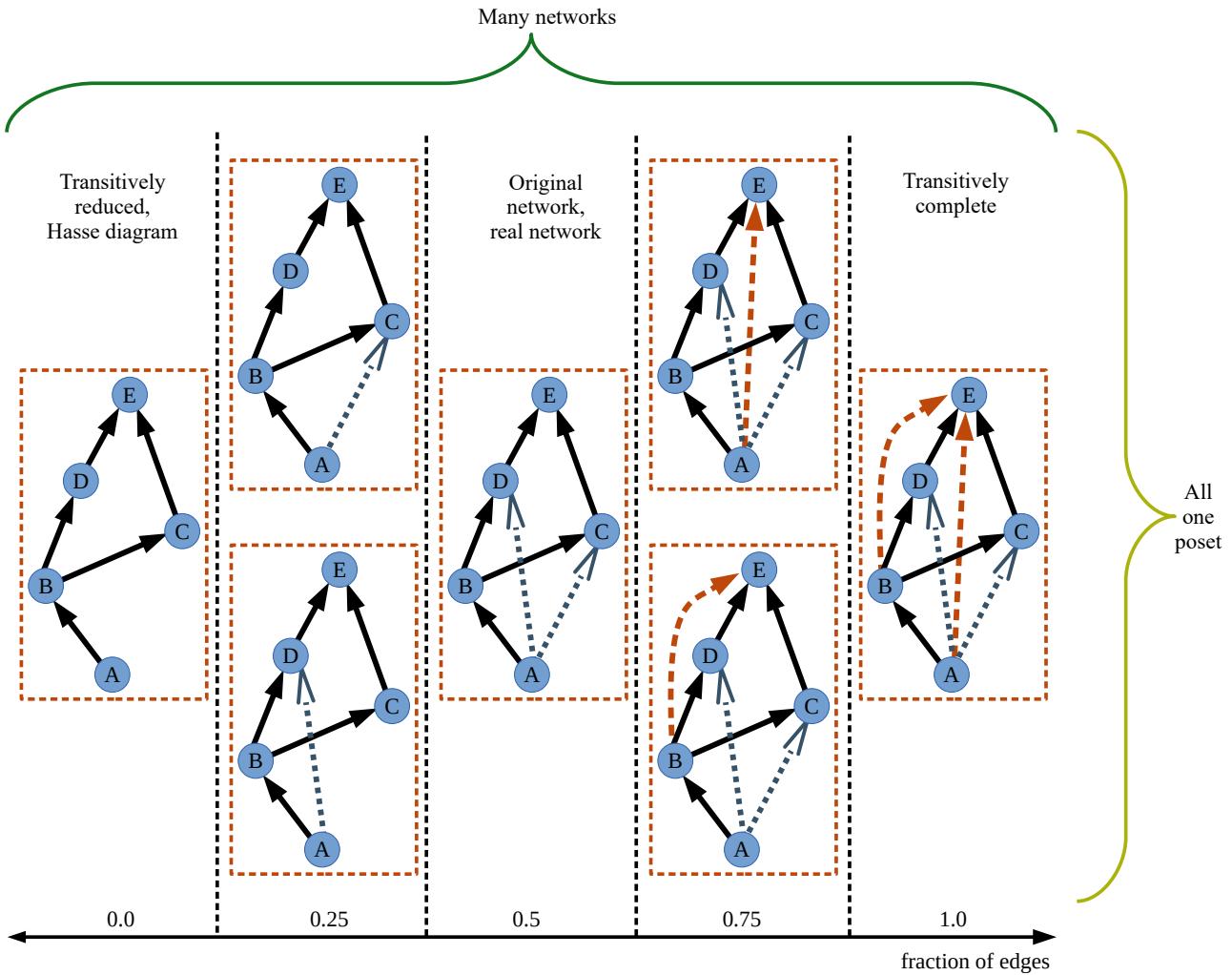


Figure 2.14: An illustration of a family of related DAGs. All these DAGs have the same transitive reduction (far left) and the same transitive completion (far right) DAG and are all equivalent to the same poset. The DAGs shown are ordered from left to right by the fraction of possible inessential edges present in that DAG. The central DAG represents the original DAG, perhaps that obtained from data. To the left are the DAGs formed by the power set of the inessential edges in the original DAG, those edges shown with a blue dotted line. To the right are the DAGs formed by the power set of the remaining inessential edges needed to reach the transitively complete graph, those edges shown as red dashed lines. Each family is related to one and only one poset, see section 2.9. This is the poset where the transitively reduced DAG is the Hasse diagram representation of that poset.

different numbers of edges. That is we have $\mathcal{F} = (\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_L)$ where here \mathcal{D}_ℓ contains all the edges of the original DAG \mathcal{D} that have network length ℓ or less. The last in this sequence is the original DAG \mathcal{D} as by definition the longest network length of any edge is equal to the height L of the original DAG so $\mathcal{D}_L = \mathcal{D}$. An example of this sequence is shown in Fig. 2.16 where the original graph is the transitively complete graph D_3 on the far right.

We can extend this argument about inessential edges to look at the role of inessential edges paths to see that the longest path between *any* two nodes is always made up of edges of network length 1, i.e. the longest paths only ever follow essential edges. Suppose we have a path $\mathcal{P} = (v_n)_{n=0}^L$ from v_0 to v_L then the edges in the path are $\mathcal{E}(\mathcal{P}) = \{(v_{n-1}, v_n)\}$ as defined in (1.2.5). Suppose one of the edges (v_{n-1}, v_n) has a network length $M > 1$. Then we can replace that edge with a path of length M from v_{n-1} to v_n , say $(v_{n-1}, u_1, u_2, \dots, u_{M-1}, v_n)$. This means in turn we can find a longer path, the path $(v_0, v_1, \dots, v_{n-1}, u_1, u_2, \dots, u_{M-1}, v_n, \dots, v_L)$ of

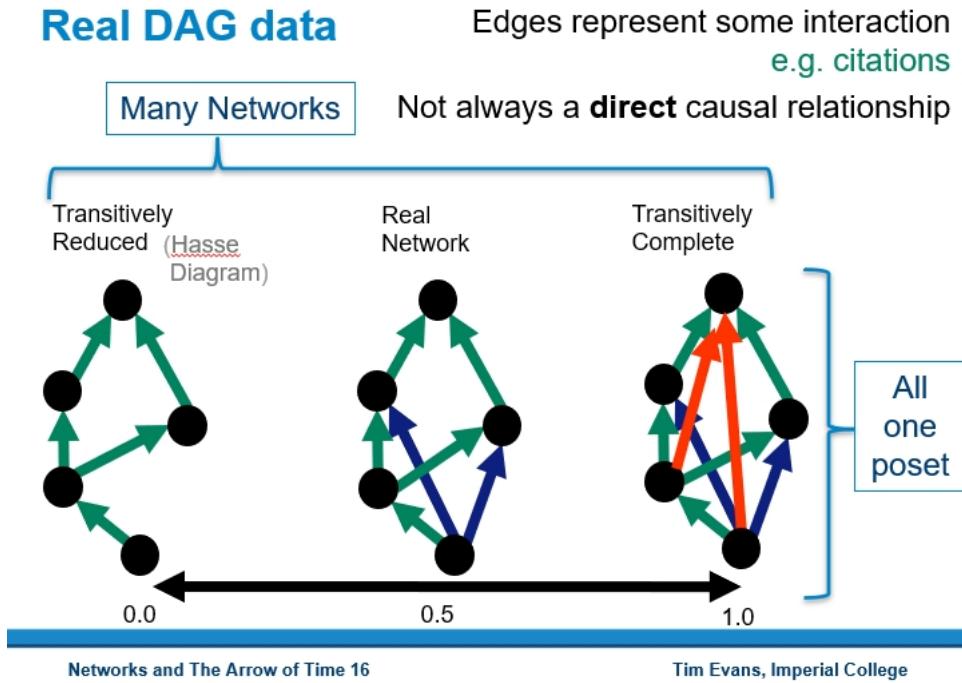


Figure 2.15: An illustration of a DAG of DAGs.

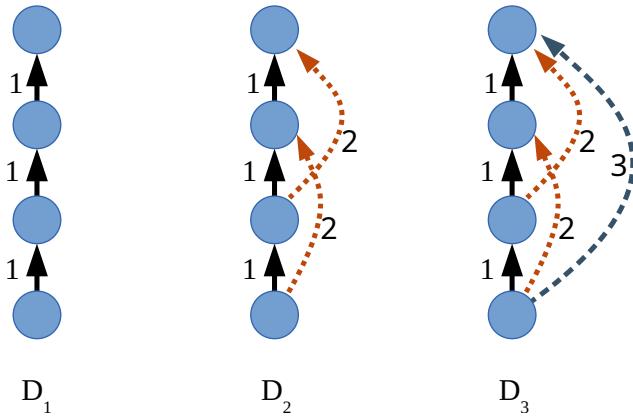


Figure 2.16: An illustration of the network length of an edge in a DAG using the same DAG as Fig. 2.6. This is equal to the longest path distance between the two nodes defining the edge. In this example the edges are labelled by their network length. All the edges of network length 1 are shown as continuous straight black lines and these are the essential edges. The remaining edges are the inessential edges shown as curved dashed lines: two edges of network length 2 in red and the single line of network length 3 in blue. The network of the far left (right) is the transitively reduced (transitively complete) graph of all three graphs.

length ($L + M$), from v_0 to v_L . Hence any path \mathcal{P} which has an inessential edge of network length two or more cannot be a longest path. Equally, the longest path must only be made from essential edges of network length one.

The converse is not true however. That is any path made only essential edges need *not* be the longest path between the initial and final nodes in the path. To see this we need only find a counter example and one is shown in Fig. 2.17. Another example is the path (A, B, F, H, J) in Fig. 2.16 since A to J can be linked of a path length 5 as shown in Fig. 2.6.

The fact that all longest paths are made up of essential edges shows that the distance $d_{vu}^{(lp)}$ measured using the length of the longest path from any node u to any node v remains unchanged under transitive reduction or transitive completion. More generally any of the DAGs in the DAG-of-DAGs family, those DAGs that share the same TR and TC DAG and illustrated in Fig. 2.13, will have the same longest path distances between the same pair of nodes. This

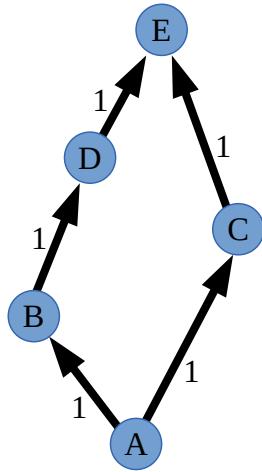


Figure 2.17: An illustration of a network with a path made up only of essential edges that is not a longest path. All the edges are essential edges, as indicated by the network length of 1 shown next to each edge. However the path (A, C, E) only has essential edges but is not the longest path from node A to node E . The path (A, B, D, E) is the longest path from node A to node E and as the longest path, it must only have essential edges which is what we see in this example. Note this also shows that even in a transitively reduced DAG, the longest and shortest paths between two nodes, here A to E , need not be of the same length.

in turn tells us the height and depth of any one node is also the same for all DAGs in the DAG-of-DAGs. In a practical sense, if we think of inessential edges as somehow ephemeral in a practical example, then all of these features based on longest paths are invariant under the presence or otherwise of such edges. It is a further reason why using the longest paths as the key measure on a DAG may be appropriate in many problems.

(T)Shortest paths in TR graphs? An interval with just two paths between source and sink is a TR graph but as the paths can be of different lengths this shows that the shortest path from source to sink is not the same length as the longest-path. So the betweenness of nodes will depend on SP/LP. **(end T)**

2.7.1 Beyond Transitive Reduction

Look at the family of a DAG. Can break a DAG or poset into layers based on network length of edges.

$$[\mathbf{A}(\ell)]_{uv} = \delta(d^{(lp)}(u, v), \ell) \quad (2.7.3)$$

Note that $\mathbf{A}(\ell = 0) = \mathbf{1}$, $\mathbf{A}(\ell = 1) = \mathbf{A}^{(tr)}$, $\sum_{\ell=1}^H \mathbf{A}(\ell) = \mathbf{A}$.

See (Zhao and Feng, 2022). Reducing past set to a tree using greedy/random algorithm.

2.8 The Adjacency Matrix of a DAG

(T)Could peel off from here into new chapter on DAGs and other structures. Also see triangular matrix. **(end T)**

There is a close relationship between a DAG and what are known as TRIANGULAR MATRICES.

Lemma 2.6. *The adj mat is always tri*

First recall that the nodes of any DAG can be put into a total order. That is we can associate an index $i(v)$ to every node v such that for any edge from node u to node v present in the DAG then the index is such that $i(u) < i(v)$. The converse need not be true, just because $i(u) < i(v)$ does not mean there is an edge between the two nodes nor even that there is a path.

We will now assume that the indices are integers running from 1 to N . If we use these indices to represent the nodes then let node u be represented by index i and node v by the index j . The existence of an edge from u to v is equivalent to setting the entry in the i -th column

and j -th row entry of the adjacency matrix \mathbf{A} to be one, $A_{ji} = 1$. Note that we are using the convention¹⁸ set out in Fig. 1.4 where the order of the indices is reversed compared to the way we write the nodes in the notation for an edge.

However, because we have chosen our indices to respect a total order, we must also have in this case that $i < j$. In addition, we know there can be no edge in the reverse direction in a DAG so $A_{ji} = 0$. Finally, there are no self-edges in a DAG, as they represent trivial cycles, so $A_{ii} = 0$. So, with an appropriate labelling of the nodes in a DAG, the adjacency matrix of any DAG can be made to be what is called a STRICTLY LOWER TRIANGULAR MATRIX. An example is shown in Fig. 2.18

A note on the nomenclature here. A triangular matrix is one where all the non-zero entries are on one side of the diagonal are zero so a LOWER TRIANGULAR MATRIX has zeros below the diagonal (while in an UPPER TRIANGULAR MATRIX all the entries above the diagonal are zero). In general triangular matrices can have non-zero entries on the diagonal but that is not what we have with DAGs. So the adjacency matrix of a DAG always has zeros along the diagonal and is a triangular matrix has zeros on the diagonal it is called a STRICTLY TRIANGULAR MATRIX.

A point worth mentioning is that every singular triangular matrix gives a DAG but different triangular matrices can give a DAG with the same shape or topology. For instance see the four node DAG in Fig. 2.28. We will discuss this in more detail ion section 2.12.

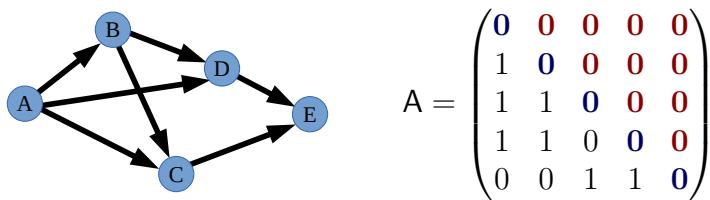


Figure 2.18: The adjacency matrix \mathbf{A} for the DAG shown on the left using the convention of Fig. 1.4. The entry A_{ij} uses indices $i, j \in \{1, 2, 3, 4, 5\}$. Suppose we let i_v denote the integer index for node v . The convention we use is that if $u \prec v$ then we must have that $i_u < i_v$, that if there is a path from the node u node to the node v then node u has the lower index, $i_u < i_v$. This map from node to integer index defines a total order so there is often more than way to do this. In this case we have chosen: $i_A = 1$, $i_B = 2$, $i_C = 3$, $i_D = 4$ and $i_E = 5$. There are no self-loops in a DAG so the diagonal of the matrix is zero making the adjacency matrix of the DAG a strictly lower triangular matrix.

The adjacency matrix of a DAG is also NILPOTENT. That is $\mathbf{A}^n = \mathbf{0}$ for all integers $n > H$ for some value H . In our case, the maximum power of the adjacency matrix that is not equal to the zero matrix is the height H of the DAG. This reflects the fact that for any unweighted network (i.e. entries A_{ij} in the adjacency matrix are all one or zero) we have that $[\mathbf{A}^n]_{ij}$ is the number of paths from node with index i to node of index j (see section ??). So since no path in a DAG can have length longer than the height, the height H is the cutoff value for the power in the nilpotent definition.

Nilpotent matrices, and so the adjacency matrices of DAGs, have several related properties: the determinant is zero, the trace is zero, all the eigenvalues are all zero and they cannot be inverted. In network science, the adjacency of most networks is usually much better behaved, having an inverse for instance, and many techniques rely upon the standard linear algebra methods such as eigenvector decomposition. None of that works here so once again, the acyclic properties of the DAG mean that if we want to respect the order encoded in our DAG then we must find new ways to analyse a DAG.

¹⁸This convention is a matter of choice. We could set $A_{ij} = 1$ for this edge. Everything would work out in a similar way except some matrices here would appear transposed, lower triangular matrices in the main text would become upper triangular matrices.

Luckily, this same acyclic nature is why we have a triangular matrix and such matrices have a whole set of useful properties of their own. They are often encountered in numerical algorithms used to perform various matrix operations. Triangular matrices with non-zero diagonal values have special properties as you can invert them using a simple iterative scheme. As we noted, DAGs have adjacency matrices with zeros on their diagonal and they are not invertible. However, in the analysis of generic networks, there are many other matrices analysed, related to the adjacency matrix but with different values, often relating to different processes. One of the best known examples is the network Laplacian \mathbf{L} where for an undirected network with no self-loops we have $\mathbf{L} = \mathbf{D} - \mathbf{A}$ with $D_{ij} = \delta_{ij}k_i$, $k_i = \sum_j A_{ij}$ being the degree of node of index i . So in this spirit we will look at one example of a matrix \mathbf{Z} which is sum of the adjacency matrix of a DAG and the unit matrix, so

$$\mathbf{Z} = \mathbf{1} + \mathbf{A}. \quad (2.8.1)$$

One way to use this matrix is to suggest that we have measured values \mathbf{f} on the nodes of a DAG, so f_i is the value on a node with index i . Suppose we write

$$\mathbf{f} = \mathbf{Zg}. \quad (2.8.2)$$

One way to interpret this is to think of the value f_i as representing some global large scale value we can measure for the node with index i . The idea is that these global values are made of sums of microscopic or individual values g_j but the only nodes that contribute their individual values g_j to the large scale value f_i of node i are the ones which are lower down the order, i.e. for $i \geq j$. For instance, perhaps we can measure the impact f_i of a technological development i . Our theory might be that the impact we measure is really a sum of the true value of previous developments plus the intrinsic value of development i . A DAG would capture the relation of the development i to past developments. The goal is to reveal the true added-value, the intrinsic value g_j of each development j , rather than the measured f_j that is also a measure of the impact of previous developments that led to i , “building on the shoulder of giants”.

To solve this problem, to find \mathbf{g} given \mathbf{f} , is a simple matter of inverting the matrix \mathbf{Z} . In fact, this structure is a well-known in mathematics and the inverse of \mathbf{Z} for certain types of DAGs known as posets (see section 2.9) is known as the MÖBIUS INVERSION. So we shall denote the inverse as $\mathbf{M} = (\mathbf{Z})^{-1}$. To find this Möbius inversion for any strictly lower triangular matrix \mathbf{A} , i.e for any DAG, first note that $\mathbf{MZ} = \mathbf{1}$. Then we can rearrange to see that $\mathbf{M}(\mathbf{Z} - \mathbf{1}) + \mathbf{M} = \mathbf{1}$ and so we have that

$$\mathbf{M} = (\mathbf{Z})^{-1} = \mathbf{1} - \mathbf{MA} \quad (2.8.3)$$

$$M_{ji} = \begin{cases} 1 & \text{if } j = i \\ -\sum_k M_{jk}A_{ki} & \text{if } j > i \\ 0 & \text{otherwise, i.e. } j < i \end{cases}. \quad (2.8.4)$$

$$= \begin{cases} 1 & \text{if } j = i \\ -\sum_{k=i+1}^j M_{jk}A_{ki} & \text{if } j > i \\ 0 & \text{otherwise, i.e. } j < i \end{cases}. \quad (2.8.5)$$

To use this iteratively we can look at each row j independently and in any order as the M_{ji} only depends on values M_{jk} from the same row j . So suppose we fix the row to be j . Now we have to work through the columns in reverse numerical order. For each row i the strictly lower diagonal nature of the adjacency matrix means that we only need to know the M_{jk} value from columns k with index larger than i we are trying to find. So if we start with the largest column index and work down, we will always have the M_{jk} values we need. The fact that we

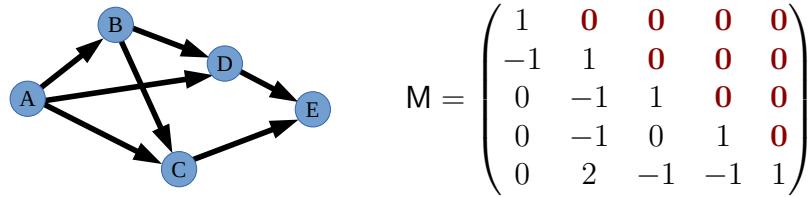


Figure 2.19: The Möbius inversion M , the inverse of $Z = \mathbf{1} + A$ where adjacency matrix A if for the DAG shown on the left using the convention of Fig. 1.4. In this case we have chosen: $i_A = 1$, $i_B = 2$, $i_C = 3$, $i_D = 4$ and $i_E = 5$.

have constructed an explicit solution for M also shows that the inverse of Z , namely the Möbius inversion M , always exists. An example is shown in Fig. 2.19.

The linear algebra used to derive (2.8.4) is just a trivial rearrangement of the definition of an inverse. If these were ordinary invertible matrices, this would be of little use or interest. It is only because the matrices are triangular than this particular rearrangement reveals the ability to calculate an inverse using a simple recursive relationship.

Let us give a simple example of the use the Möbius inversion in the context of citation networks. In this case we will assume that the edges go from newer papers to older papers. The normal measure of the impact of a paper with index j is the citation count $k_j^{(\text{in})}$. If we imagine that these values form our \mathbf{f} in (2.8.2) then we are saying that the impact $f_j = k_j^{(\text{in})}$ we measure for paper index j is made up of a sum of the the intrinsic values g_i of papers which are cite j . That is $k_j^{(\text{in})} = 1 + \sum_{i=0}^{j-1} A_{ji} g_i$. We now use the Möbius inversion to find the intrinsic value of each paper. For the example of Fig. 2.19 we find that $\mathbf{g} = (0, 1, 1, 1, 0)$ where the first value is for A and the last for E . It is hard to see an obvious pattern. In the case of citations there is a simple solution. Suppose each paper has an intrinsic value of $g_i = 1$. Then from $\mathbf{f} = Z\mathbf{g}$ (2.8.2) we see that $f_j = \sum_i [\mathbf{1} + A]_{ji} = 1 + k_j^{(\text{in})}$. That is the typical measure of the impact of a paper, the citation count $k_j^{(\text{in})}$, is simply counting the number of edges if every paper contributes the value 1, our intrinsic values g_i . IN the form used for the Möbius inversion we also add 1 to every paper j coming from its own intrinsic value g_j . So the standard measure of impact of a paper in bibliometrics fits into this structure simply by assigning each paper to have the same intrinsic value in which case the impact value we find is¹⁹ $f_j = 1 + k_j^{(\text{in})}$.

(T) What is we look at functions? $Z(A)$ where $Z(0) = 1$. (end T) (T) What if we split A into parts e.g. $A = \sum_{\ell=1}^H A^{(\ell)}$, $A^{(\ell=0)} = \mathbf{1}$. (end T)

Incidently while the matrix $Z = \mathbf{1} + A$ is invertible unlike the adjacency matrix A for a DAG, the sense of order in the DAG still limits the use of linear algebra. This can be seen when we realise the matrix Z for a weakly connected DAG has only one distinct eigenvalue $\lambda = 1$ and only one right eigenvector (first entry one, all other entries zero) and one left eigenvector (last entry one, all other entries zero).

2.9 Posets

(T) See brilliant.org or Wikipedia for outlines on posets. (end T)

Let us leave DAGs to one side for a moment and look at a well known mathematical structure often found in sets, a PARTIAL ORDER. When a set has this structure it is known as a

¹⁹Interestingly it is common to work with $1 + k_j^{(\text{in})}$ as a measure of impact whenever logarithms of the citation count are used. Since citation counts have fat tails, this is common, The problem is that zero citation counts are common and cause problems in analysis. So $\ln(k_i^{(\text{in})} + 1)$ is often used as the variable. This was exactly what was used by Price (?) in some of the earliest bibliometrics work.

PARTIALLY ORDERED SET or POSET for short. Posets are defined in terms of a binary relation that expresses a sense of order between some pairs of the elements in the set. That is for some elements x and y in some set \mathcal{X} (known as the GROUND SET) we have a sense that x comes before y , hence the ‘order’ in the name. We will represent this order as $x \preceq y$ but note at this stage this symbol here does not refer to the existence of some path in some DAG. We say that x and y are COMPARABLE if there is a relation between the two elements. It makes sense to allow an element to be always comparable to itself so in a poset we will always have that $x \preceq x$ for any $x \in \mathcal{X}$.

However the ‘partial’ refers to the fact that sometimes we do not know if x comes before y or if y comes before x , perhaps this is not even defined for this pair of elements. We say in this case that x and y are INCOMPARABLE. We represent this case by having no such relationship here, so $x \not\preceq y$ and $y \not\preceq x$, but normally we just don’t write this non-relationship down for such an incomparable pair of elements.

However, this is not enough. To be a useful sense of order we cannot have $x \preceq y$, $y \preceq z$ and $z \preceq x$. In this case we would not know which object is top of the order and so we would have no order. For instance if three teams X , Y , and Z played each other once and if Y beat X , Z beat Y but X beat Z then it is hard to determine a winner, i.e. who is top of some ordering of the teams. So poset also requires that if $x \prec y$ and $y \prec z$ then we must have that $x \prec z$, a property known as TRANSITIVITY. In fact a poset is defined formally in terms of three properties as follows.

(T) \mathcal{X} or \mathcal{P} for the set elements? **(end T)**

A PARTIAL ORDER \preceq on a set \mathcal{P} is a binary relation which satisfies three conditions (?)

$$\text{reflexive: } x \preceq x, \quad (2.9.1)$$

$$\text{antisymmetry: } x \preceq y \text{ and } y \preceq x \text{ iff } x = y, \quad (2.9.2)$$

$$\text{transitive: if } x \preceq y \text{ and } y \preceq z \text{ then } x \preceq z. \quad (2.9.3)$$

where these hold for any $x, y, z \in \mathcal{P}$. Note that $y \succeq x$ is an alternative notation for $x \preceq y$. An obvious example of a partial order is the usual “less than and equal to” relation \leq on a set of real numbers so that \preceq is \leq in this case²⁰

It will be useful sometimes to define a related relationship between two objects in the set, \prec , which is the same as the binary relation of a partial order \preceq except that we exclude the reflexive case where $x = y$. That is

$$x \prec y \quad \text{if} \quad x \preceq y \text{ and } x \neq y. \quad (2.9.4)$$

Again, $y \succ x$ means the same as $x \prec y$. Clearly for integers and real numbers $<$ plays the role of \prec while \leq plays the role of the partial order relation \preceq . The difference is that we can compare all real numbers using $<$ and \leq but the partial order may not be defined for some pairs in the set.

Just like a DAG, we can define MINIMAL ELEMENTS $s \in \mathcal{S}$ which are elements where there is no element that comes before a source element in the partial order. That is for any source $s \in \mathcal{S}$ there is no $y \in \mathcal{X}$ s.t. $y \preceq s$. Put another way for source elements $s \in \mathcal{S} \subseteq \mathcal{X}$ then for all elements $y \in \mathcal{X}$ either $y \succeq s$ or s and y are incomparable. We also have the reverse situation where we have MAXIMAL ELEMENTS $t \in \mathcal{T}$ is $x \in \mathcal{X}$ such that for all $y \in \mathcal{X}$ either $y \succeq x$ or x and y are incomparable. Alternatively, a minimal element is *not* larger than any other element, i.e. there is no $y \in \mathcal{X}$ s.t. $x \succeq y$.

Let us illustrate these ideas with some simple examples.

²⁰Confusingly, the greater than or equal to on a set of real numbers also gives a distinct but related partial order where now \geq is our \preceq . That is $x \preceq y$ if $x \geq y$ for x and y integer satisfies all the criteria. Posets always have this sort of reversal symmetry.

Example 2.9.1 Integer divisibility

Consider a poset $\mathcal{P}(\mathcal{X}, \preceq)$ where the ground set is $\mathcal{X} = \{2, 3, 4, 5, 6, 7, 8, 9\}$ and partial order is divisibility, i.e. $x \prec y$ if x divides y . We assume that $x \preceq x$ is always true (reflexivity) on the grounds that any integer is divisible by itself. We quickly see that the prime numbers, here $\{2, 3, 5, 7\}$ are the minimal elements as they have no divisors. Equally, the maximal elements are all the numbers 5 to 9 as they are too big to be a divisor of anything in this poset \mathcal{P} . The non-trivial relations are $2 \preceq 4, 2 \preceq 6, 4 \preceq 8, 3 \preceq 6$ and $3 \preceq 9$ along with one relation we must have by transitivity which we can check is indeed true, namely $2 \preceq 8$. Note 5 and 7 are not comparable to any other numbers in this poset, they can only be compared to themselves $5 \preceq 5$ and $7 \preceq 7$. Equally many other pairs are not comparable for example $6 \not\preceq 9$ nor $9 \not\preceq 6$ so 6 and 9 are incomparable.

Two of the most important examples of posets are given by set inclusion and set partition. Many posets defined in terms of other elements are equivalent to one of these two abstract examples.

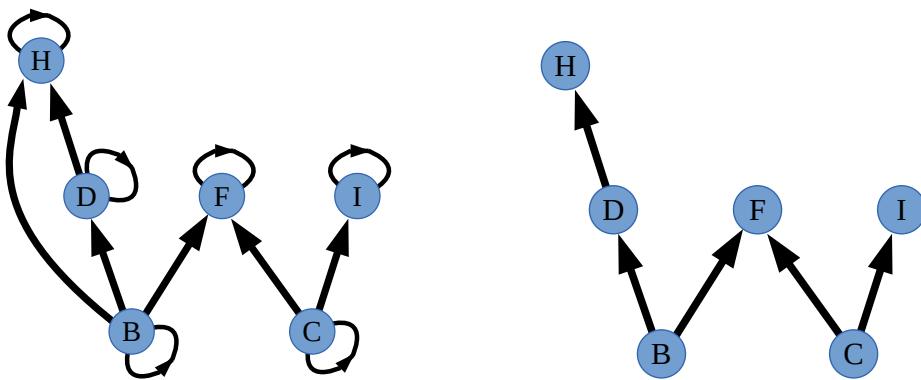


Figure 2.20: On the left is a directed network representation of a poset. This represents the poset of integer divisibility defined in Example 2.9.1 but now limited to $\mathcal{P} = \{2, 3, 4, 6, 8, 9\}$ where $B \equiv 2$, $C \equiv 3$, $D \equiv 4$, $F \equiv 6$, $H \equiv 8$ and $I \equiv 9$. Here a directed edge (u, v) is present if there is v/u is an integer. Note that we have to include self-loops, edges (v, v) for all nodes v , in order to represent the reflexivity relation $v \preceq v$ which is a necessary property of a poset. It is because of these self-loops that this directed network is not a DAG. However, if we dropped the self-loops then we would have a transitively complete DAG. The transitive reduction of this DAG still encodes sufficient information to uniquely define the directed network and hence the poset. So on the left we show the DAG found after first removing self loops and then imposing transitive reduction and this is the Hasse diagram which completely defines the poset.

Example 2.9.2 Set Inclusion

The elements x of the poset \mathcal{P} are sets and the partial order is set inclusion. So $\mathcal{P} = (\mathcal{X}, \subseteq)$ where

$$x \preceq y \text{ iff } x \subseteq y \quad \forall x, y \in \mathcal{X}. \quad (2.9.5)$$

For instance if $x = \{A, B, C\}$, $y = \{A, B\}$ and $z = \{B, C\}$, then $y \preceq x$ since $y \subseteq x$, $z \preceq x$ since $z \subseteq x$, but y and z are incomparable as there is no relation between this pair since $y \not\subseteq z$ and $z \not\subseteq y$.

One special example of a set inclusion poset comes from the power set of a set \mathcal{X} which is the set of sets containing all subsets of \mathcal{X} including the improper subsets \mathcal{X} itself and the empty set \emptyset . For example consider a set $\mathcal{C} = \{R, G, B\}$ containing the three primary colours, denoted here by “R” (red), “G” (green) and “B” (blue). The power set of \mathcal{C} , say $\mathcal{C}_{\text{power}}$, contains the three distinct sets containing pairs of colours: yellow $\{R, G\}$, cyan $\{G, B\}$ and purple $\{R, B\}$. The power set also has the three sets each containing just a single primary colour: red $\{R\}$, green $\{G\}$ red and blue $\{B\}$. You must always include the empty set (black) \emptyset and the original set, (white) $\{R, G, B\}$. So the elements of this power set of colours $\mathcal{W}_{\text{power}}$ are

$$\mathcal{C}_{\text{power}} = \{\{R, G, B\}, \{R, G\}, \{G, B\}, \{R, B\}, \{R\}, \{G\}, \{B\}, \emptyset\}. \quad (2.9.6)$$

Using set inclusion (2.9.5) then gives us a natural poset here $\mathcal{P} = (\mathcal{W}_{\text{power}}, \subseteq)$.

Example 2.9.3 Set Partition

The elements X of the ground set \mathcal{X} of this poset \mathcal{P} are different partitions of some set \mathcal{Y} . A partition is defined in (A.1). The partial order is one partition is a made of sub-partitions of another. It is simplest to give an example, Suppose we have $\mathcal{Y} = a, b, c$ then this has 4 different partitions:

$$X_1 = \{\mathcal{Y}\} = \{\{a, b, c\}\}, \quad (2.9.7)$$

$$X_{2a} = \{\mathcal{Y}_a, \mathcal{Y}_{bc}\} = \{\{a\}, \{b, c\}\}, X_{2b} = \{\mathcal{Y}_b, \mathcal{Y}_{ac}\} = \{\{b\}, \{a, c\}\}, X_{2c} = \{\mathcal{Y}_c, \mathcal{Y}_{ab}\} = \{\{c\}, \{a, b\}\}, \quad (2.9.8)$$

$$X_3 = \{\mathcal{Y}_a, \mathcal{Y}_b, \mathcal{Y}_c\} = \{\{a\}, \{b\}, \{c\}\}. \quad (2.9.9)$$

The idea is that $X_3 \preceq X_{2b}$ since the \mathcal{Y}_a and \mathcal{Y}_c elements of X_3 are themselves a partition of the $\mathcal{Y}_{ac} = \{a, c\}$ element of X_{2b} , i.e. X_3 is a subpartition of X_{2b} .

Finally we can always define at least one TOTAL ORDER (also known as a LINEAR ORDER) on a poset. This is where we place the elements of the poset in a sequence $\mathcal{T} = (x_i)_{i=1}^N$ such that if $x_i \preceq x_j$ then the index i must be less than the index j , $i < j$. There are two corollaries. First, if $i < j$ then x_j cannot precede x_i , so $x_j \not\preceq x_i$. The second is that if $i < j$ that does not force x_i to be related to x_j by the partial order, they could also be unrelated.

All posets have at least one total order but equally there can be more than one. We will show this by linking the the total order in a poset to the total order of the associated DAG in section 2.9.1.

2.9.1 Relation of DAGs to Partially Ordered Sets

(T)Self-loops, the equality in the partial order. **(end T)**

The key idea behind a poset or DAG is that the binary relationships, encoded by the partial order \preceq and the directed edges respectively, represent *dependence* and *order* between pairs of

nodes, rather than similarity of nodes.

A DAG always defines a PARTIAL ORDER \preceq on the set of nodes \mathcal{V} where we the partial order to hold whenever there is a path between two nodes, that is a path from u to v in the DAG means $u \preceq v$ in the poset and vice versa. Note we require this must include trivial paths of length zero, the path (u) from u to u , so that we have reflexivity $u \preceq u$.

In general, where two nodes are comparable there need not be an edge, that is $u \preceq v$ does not imply $(u, v) \in \mathcal{E}$, we only need a path to be present. Only if the DAG is transitively complete is there an edge for every comparable pair of nodes in the poset.

Also the family of DAGs discussed above, those with the same transitively complete DAG and so also with the same transitively reduced DAG, are all equivalent to the same poset. The reverse is true, each poset corresponds to one such family of DAGs. In that sense, a DAG contains more information than a poset as many DAGs mapped to one poset.

One corollary of this is that properties of the family of DAGs linked to one poset which are the same for all the DAGs are really properties of the underlying poset. For instance the length of the longest path between nodes is invariant. By way of contrast, the length of the shortest path between any two nodes in one family of DAGs is does change.

Finally, this suggests that we can use the transitively reduced DAG as a graphical representation of any poset and in this context it is known as a HASSE DIAGRAM. It is unique and distinct for each poset and has the minimum number of graphical elements needed to convey the information needed to define the DAG.

We can use this DAG-poset relation to link the concepts of the total order in the DAG to the total order in the corresponding poset.

(T)TODO (end T)

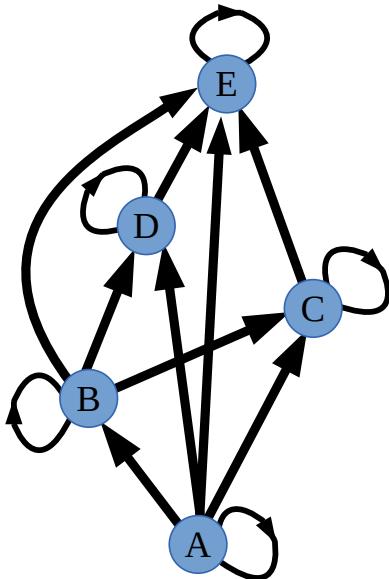


Figure 2.21: This is a directed network representation of the poset defined by the paths between nodes in the DAG of Fig. 2.1a. Here an directed edge (u, v) is present if there is a path from u to v in the DAG of Fig. 2.1a, i.e. if $u \preceq v$. Note that we have to include self-loops, edges (v, v) for all nodes v , in order to represent the trivial paths of length 0 in the DAG and the reflexivity relation $v \preceq v$ which is a necessary property of a poset. It is because of these self-loops that this directed network is not a DAG. However, if we dropped the self-loops then we would have the transitively complete DAG of Fig. 2.1a.

Note that if the adjacency matrix of this directed network in Fig. 2.20 (without any edge weights) is \mathbf{A} then we have $\mathbf{A} = \mathbf{1} + \mathbf{A}^{(tc)}$ where $\mathbf{A}^{(tc)}$ is the adjacency matrix of a transitively complete DAG.

2.10 Chains

It is sometimes useful to define a slight generalisation of a path known as a CHAIN. This is a sequence of distinct nodes $\mathcal{C} = (v_0, v_1, \dots, v_L)$ where consecutive nodes on a chain are connected

by a path in the appropriate direction. That is

$$\mathcal{C} = (v_0, v_1, \dots, v_L) \quad \text{where} \quad v_{n-1} \prec v_n \quad \text{for } n = \{1, 2, \dots, L\}. \quad (2.10.1)$$

Note we use the operator ‘ \prec ’ rather than the full partial order ‘ \preceq ’ in order to exclude the possibility that $v_{n-1} = v_n$. The first element of a chain v_0 is called the MINIMAL ELEMENT and the last v_L is the MAXIMAL ELEMENT. However unlike the nodes on a path, the consecutive nodes on a chain need not be linked by an edge, a path of length 1, they can be linked by longer paths. So in general a chain need not be a path though a path is always a chain. In fact we can always find at least one path (and often more) where the chain is a subsequence of the nodes in that path.

For instance in Fig. 2.6 the sequence $\mathcal{C}_A(1, 5, 7)$ is a chain with minimal element 1 and maximal element 7. This sequence is not path as there is no edge between nodes 1 and 5. However it is a valid chain as the nodes 1 and 5 are connected by three distinct paths, e.g. (1, 2, 5), while 5 and 7 are directly connected by an edge. Another way to see this is that all the nodes of \mathcal{C}_A can be seen as lying on a single path; any one of the paths (1, 2, 5, 7), (1, 3, 5, 7) or (1, 2, 3, 5, 7) satisfy this criteria.

Chains are usually encountered in the theory of posets. For instance another definition of a chain in the language of posets is that a chain is a sequence of elements which are all comparable and so form what is called a total order. Since every DAG defines a natural poset (see section 2.9.1) the properties and theorems of chains can be translated into the language of DAGs.

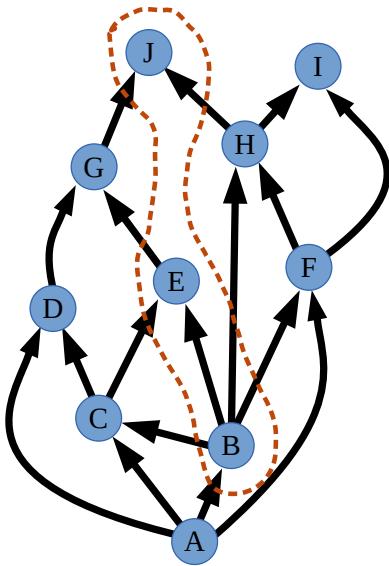


Figure 2.22: An illustration of the properties of a chain using the DAG of Fig. 2.3. Note the chain shown within the red dashed line, the node sequence (B, E, J) , is not a path though it is a subsequence of the path (B, E, G, J) .

Clearly any subset of nodes in a chain also form a chain in their own right. After all, this subset of nodes lie on the same path used to define the original chain.

The discussion of chains leads us naturally to think about how big, how long, a chain can be. Clearly, we can start with the path used to define a chain and then all the nodes on that path also define a chain. We can then try to stretch that path, adding earlier nodes that precede the original source node at the start of the path, and at the other end we can add later nodes that succeed the sink node. Eventually we cannot stretch this any more once our path is running from a source node of the DAG to a sink node of the DAG. This path defines what is called a MAXIMAL CHAIN $\mathcal{C}^{(\max)}$, a chain which cannot be made into a bigger chain by adding any other node to it. More formally it is a chain which is not a proper subset of any other chain. Maximal chains clearly run from source to sink nodes of a DAG. However there may be different maximal chains between the same DAG source and DAG sink node and moreover, they may

be of different sizes. As there can be many source and sink nodes, there are many maximal chains and we can pick out the biggest examples.

A **MAXIMUM CHAIN** is any one of these paths, one of these maximal chains of the greatest size possible. These will run between a source and a sink node of the DAG and these are the paths whose length defines the height of the DAG. Again there may be more than one such path and these can always run between different sources and sinks. For instance in Fig. 2.6 there are three maximum chains, the three longest paths $(1, 2, 5, 7, 10)$, $(1, 2, 6, 7, 10)$ and $(1, 2, 4, 8, 9)$ using two different sink nodes, the maximal node of the chains/paths.

(T) Do we need chain covers? **(end T)**

In set theory, a cover of a set \mathcal{X} is a collection of sets whose union contains \mathcal{X} as a subset. A **CHAIN COVER** of a poset is a set of chains such that every element of the poset is at least one of the chains. That is if $\mathcal{C} = \{C_1, \dots, C_c\}$ is a chain cover of poset \mathcal{P} , then the $C_a \in \mathcal{C}$ are chains of the poset \mathcal{P} and $\forall x \in \mathcal{P} \exists C_a \in \mathcal{C}$ s.t. $x \in C_a$. Note that not all relations in the poset are encoded by the chains. That is we may be able to find $x, y \in \mathcal{P}$ s.t. $x \succ y$ yet there is no chain $C_a \in \mathcal{C}$ s.t. $x, y \in C_a$. In terms of the complete DAG that is equivalent to the poset, the chain cover is not a cover of the edges, only the nodes.

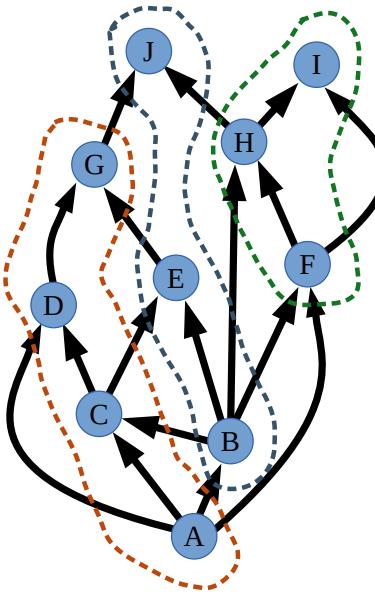


Figure 2.23: An illustration of a cover of chains using the DAG of Fig. 2.3. Each chain is the set of nodes contained within one of the three regions indicated by dashed lines. In this case two of the chains are also paths: (A, C, D, G) in the red region on the left and (F, G, H) in the green region on the right. The chain in the middle blue region, (B, E, J) is not a path since (E, J) is not connected by an edge, they are just connected by a longer path of length two (E, G, J) .

2.10.1 Antichains

21

(T)²²

An **ANTICHAIN** \mathcal{A} is a subset of a partially ordered set such that any two distinct elements in the subset are incomparable. That is if $u, v \in \mathcal{A}$ then $u \not\succ v$ and $u \not\prec v$. We can apply this to a DAG where an antichain is a set of nodes where there is no path between any of the pairs of nodes in the antichain, see Fig. 2.24 for an example.

A **MAXIMAL ANTICHAIN** is an antichain that is not a subset of any other antichain, see Fig. 2.24 for an example. In other words, if we add any element to a maximal antichain, it is not an antichain anymore. The width of the DAG and its equivalent poset is given by the size of largest antichain (which is therefore one of the maximal antichains) as illustrated in Fig. 2.24.

²¹ Adapted from Vaiva notes

²² **(T)** Width, Dilworth Theorem. Property of Poset so antichains are same for all DAGs equivalent to the same poset.

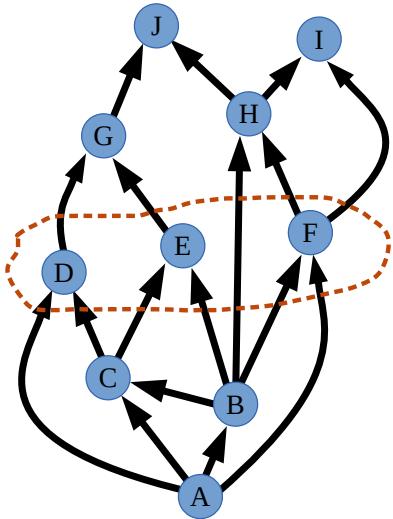


Figure 2.24: An illustration of an antichain using the DAG from Fig. 2.3. One antichain is the set of nodes $\mathcal{A} = \{D, E, F\}$. It is maximal antichain as if we add any other node, such as B or G then there is a path between some of the nodes in that set. This is the largest maximal antichain so it gives the width of this DAG as 3. We can see other antichains such as $\mathcal{A} = \{F, G\}$ which shows that nodes can be in more than one antichain and so different antichains often intersect.

Generally, a DAG is a *finite* poset (it could be infinite but in reality databases are of finite size). The width of a finite poset and, consequently, of any DAG is described by DILWORTH'S THEOREM Dilworth (1950). (Also see brilliant.org.) The following is taken from Wiki page of Dilworth's theorem.

Theorem 2.7 (Dilworth's theorem 1). *There exists an antichain \mathcal{A} , and a partition of the order into a family \mathcal{P} of chains, such that the number of chains in the partition equals the cardinality of \mathcal{A} . When this occurs, \mathcal{A} must be the largest antichain in the order, for any antichain can have at most one element from each member of \mathcal{P} . Similarly, \mathcal{P} must be the smallest family of chains into which the order can be partitioned, for any partition into chains must have at least one chain per element of \mathcal{A} . The WIDTH of the partial order is defined as the common size of \mathcal{A} and \mathcal{P} .*

Theorem 2.8 (Dilworth's theorem 2). *In any finite partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains.*

Corollary 2.9 (Erdős-Szekeres Theorem). *Let m and n be natural numbers. If \mathcal{P} is a poset with $(nm + 1)$ elements, then it has a chain of size $(n + 1)$ or an antichain of size $(m + 1)$.*

You prove this corollary by seeing that if in a poset \mathcal{P} each chain of size n or less, and every antichain is of size m or less then Dilworth's theorem implies $|\mathcal{P}| \leq mn..$

Note that this corollary shows that the size of the poset is growing as $\mathcal{P} \sim L^D$.

Theorem 2.10 (Dilworth's theorem for infinite posets). *An infinite partially ordered set has finite width (maximum size of an antichain of a poset) w if and only if it may be partitioned into w chains, but not less.*

We assume that nodes, that are in a chain, are pairwise comparable using topological sort/hierarchical structure, and nodes that are in an antichain are pairwise comparable using centrality. So, according to the Dilworth's theorem 2, if we have a partition of a poset into N_c chains, the maximum number of pairwise comparable elements using centrality is also N_c . To maximise this number, a partition of a poset into chains must also be maximal²³.

Furthermore, the minimum number of antichains into which we can partition the poset is equal to the height of it. This is Mirsky's theorem Mirsky (1971).

²³To find minimum chain partition and maximum antichain we can use Fulkerson's (1954) bipartite matching algorithm (network flows).

Theorem 2.11 (Mirsky's theorem). *For every finite partially ordered set, the height (maximum size of a chain in poset) also equals the minimum number of antichains.*

For sets of order dimension two, the two theorems coincide (a chain in the majorization ordering of points in general position in the plane is an antichain in the set of points formed by a 90 degrees rotation from the original set, and vice versa) but for more general partial orders the two theorems differ, and (as Mirsky observes) Dilworth's theorem is more difficult to prove (taken from Mirsky's theorem wiki).

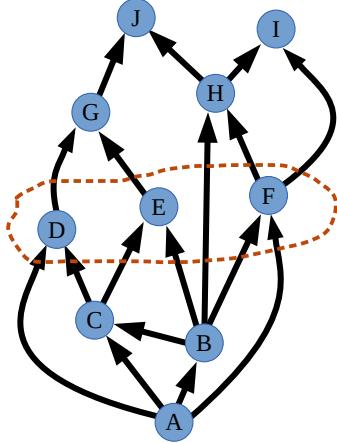


Figure 2.25: An illustration of the properties of antichains using the DAG from Fig. 2.3. **TODO.**

2.11 Trees

- The relation between DAGs and trees.
- Trees as subdags

A DIRECTED TREE (also called an ORIENTED TREE or a POLYTREE) is a DAG where the underlying undirected graph is a tree. That is if we replace the directed edges by undirected edges, we obtain an undirected graph that is connected but acyclic.

Source: Conversation with Copilot, 01/07/2024. A directed acyclic graph DAG is a graph with directed edges where there are no cycles. In contrast, a **tree** is a restricted form of a graph. Trees are also DAGs, but with the additional restriction that each child can have only one parent. In other words, a tree is a DAG without cycles, where the underlying undirected graph is connected and acyclic [2]. If you replace the directed edges of a polytree (a specific type of DAG) with undirected edges, you get an undirected tree [2].

(1) Tree (graph theory) - Wikipedia. . (2) The Algebra of Directed Acyclic Graphs - University of Cambridge. <https://www.cl.cam.ac.uk/~mpf23/papers/Algebra/dags.pdf>. (3) What's the difference between the data structure Tree and Graph?. (4) 6.1 Directed Acyclic Graphs - Rensselaer Polytechnic Institute. <https://www.cs.rpi.edu/~slotag/classes/SP20t/slides/lec06.pdf>.

For example see Fig. 2.30 which shows ...

2.12 DAGlets - small DAGs

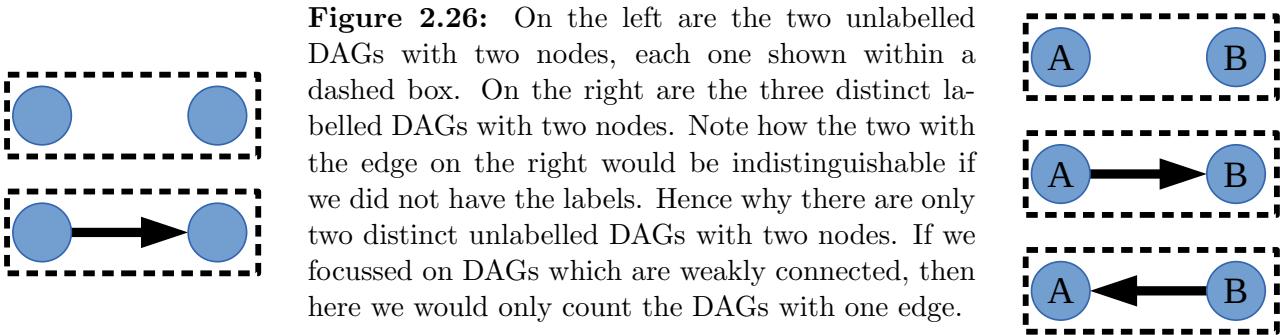
(T)Should we reserved DAGlets for the equivalent of graphlets? **(end T)**

It is often useful to think about DAGs with a small number of nodes when testing out ideas, trying to understand principles. It is also a common practice in mathematics to enumerate all

the distinct possibilities in a problem in order to organise and analyse mathematical structures in a systematic way. Finding the numbers of possible DAGs, posets and so forth is no exception as (Robinson, 1973) illustrates for DAGS and as the **OEIS** (On-Line Encyclopedia of Integer Sequences) shows in general. However finding all the DAGs with a given number of nodes is no trivial task so in practice we can only do this for small DAGs.

The first way to classify the different DAGs is to consider **LABELLED GRAPHS**. In this case, every node in the network is given a label. The label is used to indicate that each node is to be treated as distinct. This means we can tell the difference between two networks with the same shape, the same topology, but where the nodes are in a different arrangement. Consider the unlabelled DAGs of two nodes as shown in Fig. 2.26. With the nodes labelled A and B, then we only have one option with no edges. Switching the labelled on the nodes in the two node, no edge example in Fig. 2.26 produces a different visualisation but in terms of labels graphs, we consider these to be the same as the node A and B would still have the same connects, the same topology (none in this trivial case). On the other hand when we add an edge between them, we can do this in two different ways, either from *A* to *B* or vice versa. Put another way, if we were to specify these networks using an edge list and a node list, switching the labels for the first example with no edges would give you the same lists back while switching the labels in the second case would produce a slightly different edge list, that of third example.

If we let $n_N^{(L)}$ be the number of DAGs with N labelled nodes then we find that the number of unlabelled DAGs with N nodes is given by the sequence $(n_N^{(L)})_{N=0}^{\infty} = (1, 1, 3, 25, 543, 29281, \dots)$ (Robinson, 1973), listed as sequence A003024 in the OEIS (On-Line Encyclopedia of Integer Sequences). We show the three labelled DAGs with two nodes, $n_2^{(L)} = 3$ in Fig. 2.26.



We can also drop the node labels and count **UNLABELLED GRAPHS**. If we let $n_N^{(U)}$ be the number of DAGs with N labelled nodes then we find that the number of unlabelled DAGs with N nodes is given by the sequence $(n_N^{(UL)})_{N=0}^{\infty} = (1, 1, 2, 6, 31, 302, 5984, \dots)$ (Robinson, 1973), listed as sequence A003087 in the OEIS.

Perhaps more usefully, we can also focus on DAGs which have one piece, which are weakly connected. For example in the case of $N = 2$ in Fig. 2.26, only two of the three are weakly connected labelled DAGs. The number of such weakly connected labelled DAGs is given as $(n_N^{(LC)})_{N=0}^{\infty} = (0, 1, 2, 18, 446, 26430, 3596762, \dots)$ sequence A082402 in the OEIS.

- You can map the set of labelled distinct DAGs of N nodes to binary (containing only 0's or 1's) $N \times N$ matrices with positive eigenvalues. The unlabelled DAGs follow if we only consider equivalence classes of matrices under conjugation by permutations. The proof of this is simple and is laid out in McKay et al. (2004).
- See the `smalldag/smallgraph.pdf` notes and the `AllSmallNetworks.py` code for example for $N \leq 5$.

Type	OEIS/ formula	Number of Nodes					
		1	2	3	4	5	6
labelled DAGs	A003024	1	3	25	543	29281	3781503
weakly connected labelled DAGs	A082402	1	2	18	446	26430	3596762
upper strictly triangular matrices	$2^{N(N-1)/2}$	1	2	8	64	1024	32768
unlabelled DAGs	A003087	1	2	6	31	302	5984
weakly connected unlabelled DAGs	A101228	1	1	4	24	267	5647
TR weakly connected unlabelled DAGs	A000608	1	1	3	10	44	238
unlabelled directed trees	A000238	1	1	3	8	27	91

Table 2.1: The number of DAGlets of different types. The OEIS columns gives the index of the sequence on the On-Line Encyclopedia of Integer Sequences (OEIS). The second last line counts connected Hasse diagrams so this is also the number of weakly connected distinct unlabelled posets.

- Many to one strictly upper triangular binary matrices and DAGs? Numbers? Labelled vs unlabelled.

The most numerous DAGs are the labelled DAGs. In principle, for a given visualisation of a DAG with N nodes, we can label the nodes in $N!$ ways. However, because of symmetries in some DAGs, some of these $N!$ different labellings do not lead to different DAGs. Take the example in Fig. 2.27. When we switch the labels C and D we get the same labelled DAG, the same set of edges and the same adjacency matrix. This is because of a symmetry in this particular DAG. Not all DAGs have such symmetries (for instance the line of four connected nodes does not) but several will for any given N . This means that in general, the number of distinct labelled DAGs with N nodes is less than $N!$ which is the number of ways we can assign one set of labels to the nodes.

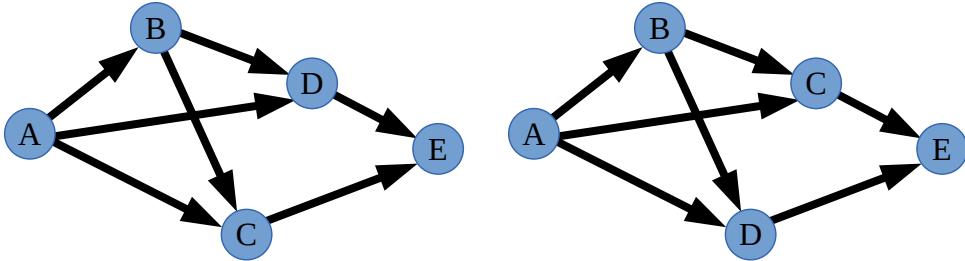


Figure 2.27: These two labelled DAGs appear to differ by the interchange of the C and D labels. However, the layout on the page has no meaning and the these two labelled DAGs are the same labelled DAG. For instance if we were to use the convention that the adjacency index corresponding to node A was 1, B was 2, etc. we would find we would get exactly the same adjacency matrix in both cases. If we were to look at the set of edges $\mathcal{E} = \{(A, B), (A, C), (A, D), (B, C), (B, D), (C, E), (D, E)\}$ then we see this is the same for both visualisations. There is a symmetry in this example DAG under interchange of the labels of the two nodes C and D .

Equally, we have noted in section ?? that every strictly upper triangular matrix corresponds to a DAG. However there are far fewer upper triangular matrices than the $N!$ permutations of labels. This is because the index $i = 1$ of the adjacency matrix must always be a source node while the node corresponding to index $i = N$ of the adjacency matrix must always be a sink. The indices of the adjacency matrix must assigned to nodes in such a way that the indices reflect a total order. That is if $u \prec v$ for any two nodes $u, v \in \mathcal{V}$ then the corresponding

indices must obey $i_u < i_v$. So the upper triangular matrices give us far fewer labelled DAGs than the $N!$ permutations of labels might suggest. For instance for $N = 2$, we have only two upper strictly triangular matrices, either $A_{12} = 0$ or $A_{12} = 1$. If we associate $i = 1$ with the label A and $i = 2$ with the label B then we see that the top two labelled DAGs of Fig. 2.26 are produced by upper strictly triangular matrices but not the third. So enumerating all the strictly upper triangular $N \times N$ matrices with zero or one entries is a relatively simple way to create all possible DAGs with N nodes but finding the unique labelled and unlabelled DAGs still poses a challenge.

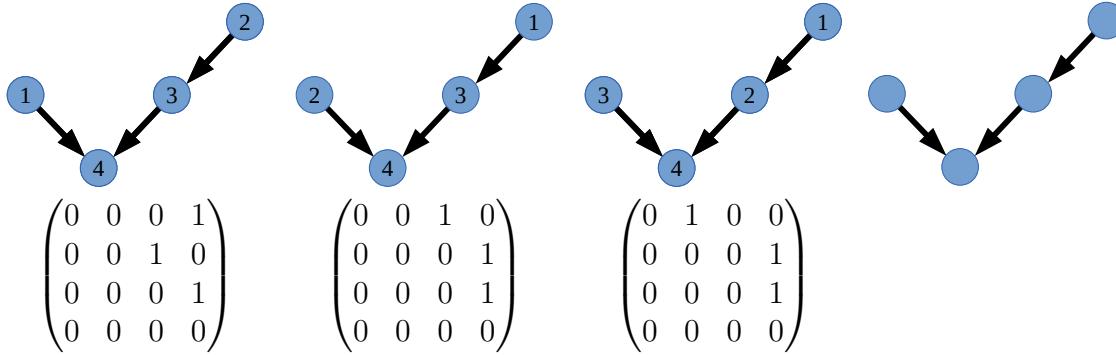


Figure 2.28: An illustration of how different upper singular triangular matrices correspond to different labelled DAGs but to the same unlabelled DAG. In this example, we show three different labelled DAGs in which the labels correspond to the index used in the adjacency matrix shown underneath. However, without these labels, so just looking at the shape, the topology, of the diagram, we can see these all correspond to the same unlabelled DAG on the right. At the same time, there are 21 other ways to label these nodes but then the adjacency matrix using these labels as indices would not be an upper triangular matrix. So there are always more distinct labelled DAGs than those labelled DAGs produced from upper strictly triangular matrices.

In terms of analysing networks in general, it is often useful to work in terms of MOTIFS in which we look at a small weakly connected subgraph, or motif, but look to see how often each motif appears in a large network. For this we are usually interested in unlabelled DAGs as it is the shape of each motif which often has some meaning. The numbers of such unlabelled weakly connected DAGs is $(n_N^{(\text{ULC})})_{N=0}^{\infty} = (0, 1, 1, 4, 24, 267, \dots)$, see OEIS sequence [A101228](#). For example for $N = 3$ see (2.29).^(T)²⁴ Of course there are fewer unlabelled DAGs than labelled DAGS as both Fig. 2.27 and Fig. 2.28 show. the set of unlabelled DAGs with one weakly connected component are much easier to handle. If we need DAGs with multiple components or labels, we can construct these variations from the smaller set unlabelled weakly connected set. For example, Fig. 2.29 summarise all the DAGs we can construct with three nodes and one component.

There is one further simplification we can make. We can further reduce the number of DAGs we consider by working with the transitively reduced diagrams. We can always find the full set of DAGs by adding in all inessential edges in all possible ways. Essentially each transitively reduced DAG represents the family of DAGs in the DAG of DAGS, the family formed from the power set of inessential edges. See Fig. 2.13 where we could use the line of four nodes to represent all eight DAGs in this family of DAGs. Any transitively reduced DAG is a Hasse diagram so at the same time this set of DAGs is in a one-to-one correspondence with different unlabelled posets. So the set of transitively-reduced weakly-connected unlabelled DAGs are the simplest way to categorize all possible DAGs. For example, in Fig. 2.29, one of our unlabelled diagrams has an inessential edge, one removed under transitive reduction, so there are only

²⁴(T)I have not found this sequence on OEIS.

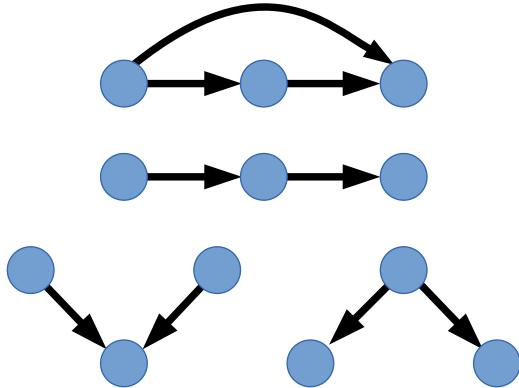


Figure 2.29: The four distinct unlabelled DAGs with three nodes and one weakly connected component. These are also the Hasse diagrams of the three possible posets within these same constraints. The DAGs are placed so the height of the DAGs decreases from 3 at the top to 1 at the bottom. Note that only the top DAG has an inessential edge so the other three are Hasse diagrams.

three distinct unlabelled DAGs with one weakly connected component. As each transitively reduced DAG is a Hasse diagram of a poset, we also see this means there are three distinct unlabelled posets of order three. In Fig. 2.30 we show all ten transitivity reduced unlabelled DAGS with four nodes and one weak component.

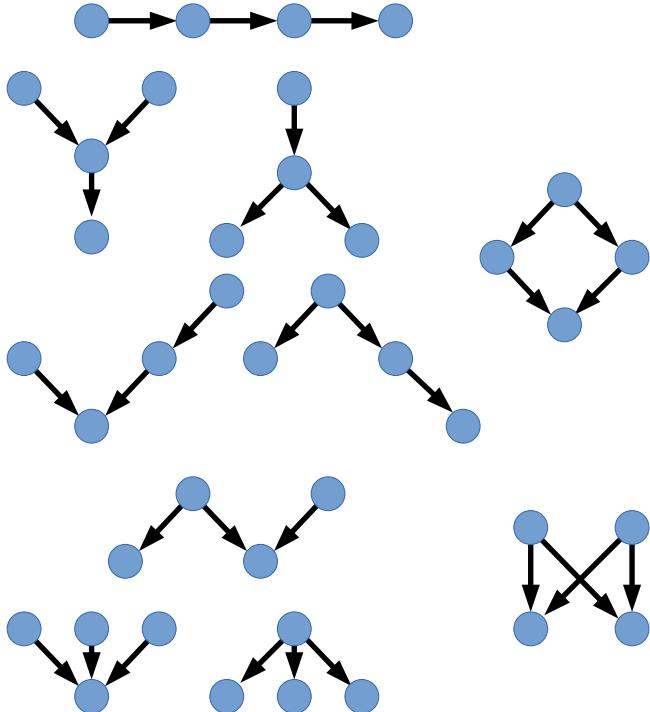


Figure 2.30: The ten distinct transitivity reduced unlabelled DAGs with four nodes and one weakly connected component. These are also the Hasse diagrams of the ten possible posets within these same constraints. They are placed so the height of the DAGs decreases from 3 at the top to 1 at the bottom. Eight DAGs have three edges but the two on the far right have four edges. So all the diagrams with eight edges are directed trees while those with four edges are not.

(TODO) UL, N , $C = 1$. TR/TC and poset numbers. (end TODO)

2.12.1 General Statistics

(T)Based on results from code in `AllSmallNetworks.py` and `network.arc_diagram.py`. (end T)

If we let $n_N^{(L)}$ be the number of DAGs with N labelled nodes then we find that this is the sequence $(n_N^{(L)})_{N=0}^{\infty} = (1, 1, 3, 25, 543, 29281, \dots)$ This is given as sequence A003024 in the OEIS (On-Line Encyclopedia of Integer Sequences). For instance for $N = 2$ we have

No. Comp	$N = 3$				$N = 4$				$N = 5$			
	USTM	UL	USTM	UL	USTM	UL	USTM	UL	USTM	UL	USTM	UL
	TR	TR	TR	TR	TR	TR	TR	TR	TR	TR	TR	TR
1	4	4	3	3	38	24	18	10	728	267	181	44
2	3	1	3	1	19	5	15	4	230	28	120	13
3	1	1	1	1	6	1	6	1	55	5	45	4
4	0	0	0	0	1	1	1	1	10	1	10	1
5	0	0	0	0	0	0	0	0	1	1	1	1
Total	8	6	7	5	64	31	40	16	1024	302	357	63

Table 2.2: The number of distinct DAGs defined by upper strictly triangular matrices (USTM) and the number of unlabelled (UL) DAGs, without transitive reduction or after transitive reduction (TR). Numbers shown broken down by the number of weakly connected components (No.Comp.) for given graph size N . The number of labelled DAGS are the same as labelled graphs.

2.13 Temporal Networks

In the literature, see for example (Holme and Saramäki, 2012, Holme, 2015, Masuda and Lambotte, 2016) and references therein, a TEMPORAL NETWORK is a network where the edges are labelled by a time. For instance the nodes could be people and the links represent communications between pairs of people, a phone call or an email with the time of that communication labelling with the link. Each pair of nodes can have many edges between them but they will all be labelled by different times as people talk to each other at many different times.

Really what is called a temporal network in the literature should be known as a *temporal edge network* as the temporal labels are given to the edges. By this logic, there is also a type of *temporal node network* where nodes are labelled by times. For instance if the nodes are documents the publication date could be the temporal label given to each node. With a time given to each node such temporal node networks have a natural representation as a DAG, where we order the edges (whatever they might represent) so they all point from say the earlier node to the later node. The order inherent in a DAG often comes from some time associated with each node so we can often think of DAGs as such temporal node networks.

We can make a more formal connection between temporal (edge) networks and DAGs using the LINE GRAPH construction. The line graph $\mathcal{L}(\mathcal{G})$ of an undirected network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a network in its own right with a node set \mathcal{V}_L and an edge set \mathcal{E}_L , so $\mathcal{L}(\mathcal{G}) = (\mathcal{V}_L, \mathcal{E}_L)$. Every edge $e \in \mathcal{V}$ in the original network \mathcal{G} is represented by a *node* in the line graph, so $\mathcal{V}_L = \mathcal{V}$. Two nodes in the line graph, say $e, f \in \mathcal{V}_L$, are connected if the edges they represent in the original graph \mathcal{G} share a common node, that is if $e \cap f \neq \emptyset$.

$$\text{If } \mathcal{G} = (\mathcal{V}, \mathcal{E}) \text{ then } \mathcal{L}(\mathcal{G}) = (\mathcal{V}_L, \mathcal{E}_L) \text{ where} \quad (2.13.1a)$$

$$\mathcal{V}_L = \mathcal{E}, \quad (2.13.1b)$$

$$\mathcal{E}_L = \{(e, f) | e, f \in \mathcal{E}, e \cap f \neq \emptyset\}. \quad (2.13.1c)$$

To make the connection with a DAG we start with a temporal (edge) network $\mathcal{G}(\tau) = (\mathcal{V}, \mathcal{E})$ where the function τ gives us the temporal label $\text{tau}(e)$, a real number, for each edge e , that is $\tau : \mathcal{E} \mapsto \mathbb{R}$. We create a line graph as before which will be an undirected network. However, now each node e in the line graph has a time label $\text{tau}(e)$ and this puts the nodes in a total order (assuming ties can be resolved satisfactorily). Using the order on the nodes, we can define a direction for each edge consistent with that order, say making edges in the line graph point

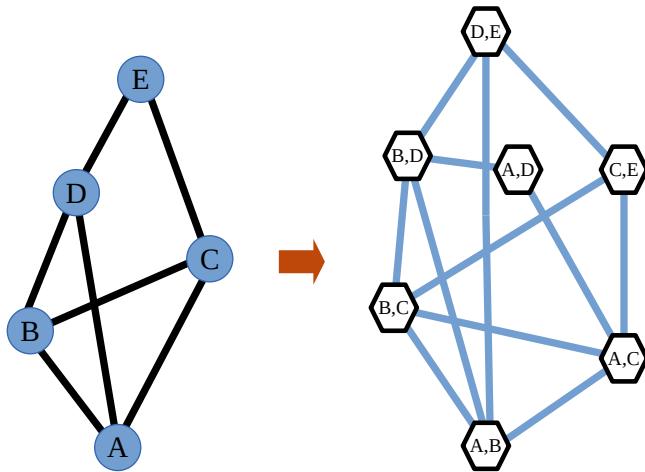


Figure 2.31: On the right is the line graph of the original undirected network shown on the left. Each node in the line graph is shown as a hexagon labelled by the edge from the left hand network that it represents. Each edge in the line graph, the blue lines in the right hand network, is generated by a node from the network on the left. For instance the edge between nodes (A, C) and (C, E) is generated by node C in the original network since C is the node in common to these two edges so that is where the two original edges meet. Each node of degree k in the original graph generate $k(k - 1)/2$ edges in the line graph.

from smaller to larger time nodes. This directed line graph must be a DAG given the total order of the nodes.**(T)**²⁵ An example is shown in Fig. 2.32.

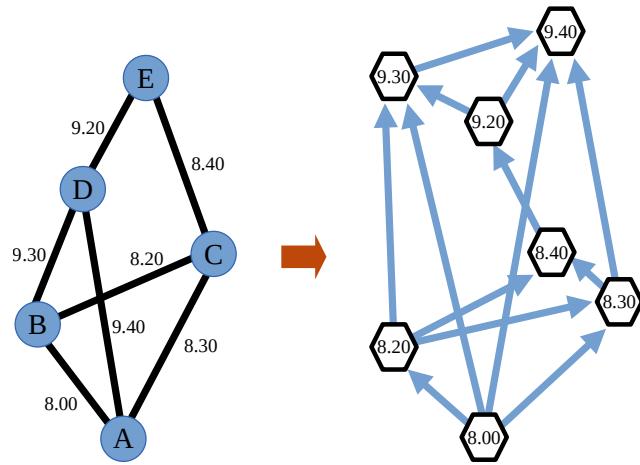


Figure 2.32: The graph on the left is the temporal graph where the labels on the edges show the times associated with that edge. On the right is the directed line graph of this temporal network but with direction added to the edges. The hexagons in on the right represent the edges from the temporal network and so these nodes carry the time label. Using that label we can add a direction to the edges in the line graph on the right. The order inherent in times ensures that these labels give a total order for the nodes in the line graph and hence the directed line graph on the right is a DAG.

2.14 Other DAG features

(T)This could be where we break the chapter. Some of these could be measures that work in directed graphs as well as DAGs. **(end T)**

²⁵**(T)**Prove this.

- You can map the set of labelled distinct DAGs of N nodes to binary (containing only 0's or 1's) $N \times N$ matrices with positive eigenvalues. The unlabelled DAGs follow if we only consider equivalence classes of matrices under conjugation by permutations. The proof of this is simple and is laid out in McKay et al 2004 McKay et al. (2004).
- The least-common-ancestor (LCA), node w , of nodes u and v in a DAG is an ancestor of both u and v where w has no descendants that are ancestors of u and v . For example see Bender et al 2001 Bender et al. (2005).
- MORAL GRAPHS $\mathcal{M} = (\mathcal{V}_M, \mathcal{E}_M)$ are undirected graphs created from the DAG $\mathcal{D} = (\mathcal{V}_D, \mathcal{E}_D)$. The node set is the same for the two graphs. All the edges in the DAG are mapped to undirected edges between the same edges pairs in the Moral Graph and in addition if two nodes in the DAG have a common child then an edge is added between these nodes in the Moral graph. The latter process is called “marrying” two nodes.

$$M : \mathcal{D} \rightarrow \mathcal{M} \text{ s.t. } \quad \mathcal{V}_M = \{M(v) | v \in \mathcal{V}_D\}$$

$$\mathcal{E}_m = \{(M(v), M(w)) = (M(w), M(v)) | (v, w) \in \mathcal{E}_D\} \cup \{(M(v), M(w)) = (M(w), M(v)) | (v, w) \in \mathcal{E}_D\}$$

The Wikipedia page on Moral Graphs has more information.

Chapter 3

Analysing DAGs

Aims and Objectives: To understand how we can analyse a DAG.

- Centrality
 - Local measures
 - Using TR/TC. Node volume measures.
 - Using one optimal path between two sites but a large range of source/target node pairs.
 - Using multiple paths between two sites and again a large range of source/target node pairs.
- Layers. Trophic analysis.

(T)Find more. (end T)

- In- or out-degree. For example the citation index in citation networks.
- Volume measures, number of ancestors and descendants.
- The ‘status-contraststatus’ measures used to analyse hierarchies in organisations Harary (1959) (see also p204-305 in (Wasserman and Faust, 1994)). Done in terms of longest- or shortest path
- The keystone index (5.3.1) of ecology. This uses all paths between two nodes. Copy from food web section. Write in generic format using left to right diffusion process. Leave food web adaptation to that later section.
- The mixed trophic impact (5.3.12) used in ecology. Again, this uses all paths between two nodes. Copy from food web section. Write in generic format using left to right diffusion process. Leave food web adaptation to that later section.
- Other ecology trophic level definitions. For instance (Gascuel et al., 2011) cite Odum and Heald, 1975; Adams et al., 1983 and give trophic level τ_i as

$$\tau_i = 1_i + \sum_j \frac{A_{ij}}{k_i} \tau_j = \sum_{u \in \mathcal{V}} \sum_{n=0}^{\infty} [(\mathbf{T}^{(in)})^n]_{vu} 1_u \quad (3.0.1)$$

where here we are going from prey j to consumer i , i.e it is the average of incoming nodes lower down the food web.

- We need to fix the direction of edges in a food web. I suggest from plants to top predators

(T)Find more. (end T)

3.1 DAG Centrality

(T)Find more. (end T)

- Local Measures
 - $k_v^{(\text{in})}$, $k_v^{(\text{out})}$ of (3.1.1) — in- or out-degree. For example the citation index in citation networks. In terms of future/past, normal/TR/TC (3.1.2) etc.
- Global measures without paths
 - $\tilde{N}_v^\pm = |\tilde{\mathcal{V}}^\pm(v)|$ of (3.1.3) and $\hat{N}_v^\pm = |\tilde{\mathcal{V}}^\pm(v)|$ of (3.1.4) — volume measures, number of ancestors and descendants, sinks/sources in past.
- Global measures with optimal paths
 - $b_v^{\min/\max}$ of (3.1.5) — betweenness
 - $c_v^{\pm,\min/\max}$ of (3.1.1) and $\sigma_v^{\pm,\min/\max}$ e:status — closeness and status-contraststatus. measures used to analyse hierarchies in organisations Harary (1959) (see also p204-305 in (Wasserman and Faust, 1994)). In terms of future/past, distances longest/shortest path or height etc.
- Global measures with all paths
 - K_v^\pm in (??) — the keystone index (5.3.1) of ecology. This uses all paths between two nodes. Copy from food web section. Write in generic format using left to right diffusion process. Leave food web adaptation to that later section.
 - S_v of (3.1.68) — the mixed trophic impact (5.3.12) used in ecology. Again, this uses all paths between two nodes. Copy from food web section. Write in generic format using left to right diffusion process. Leave food web adaptation to that later section.
 - ??? Main Path Analysis node values. Leave edge selection till later.
 - Types of path: broadcast A, stochastic random walker forwards or backwards $T^{(\text{in})}$ or $T^{(\text{out})}$, paths coming forward to the vertex v from the past, paths going backwards from v towards nodes in the past, paths going forwards from the vertex v to the future, paths going backwards from nodes in the future back towards v .

As we mentioned at the start of section 1.4.1, the larger a centrality value is for a node then the more important that node is meant to be. Given important, centrality, can be defined in many ways for different contexts, it is not surprising that there are many node centrality measures available in standard network analysis. One strategy is to drop the directionality of the edges in a DAG and run these standard centrality measures on this undirected network. If that proves useful then there is nothing wrong with that. However, here the focus is on the idea that the order of a DAG makes it distinct from all other networks. So we will look at centrality measures that respect the order in a DAG.

We will split node centrality measures for DAGs into four categories. We will first look at local centrality measures where we can find the centrality value for each node without looking at the whole structure of the DAG, such as degree. These are simple network measures seen in section 3.1.1. Then we will move on to look at measures that are based on large parts of the DAG but are not directly based on paths through the DAG, such as betweenness. These are also measures developed for ordinary networks, undirected or directed with cycles. The third category are measures where a single optimal path between pairs of nodes is used to give a DAG node centrality measure that probes the whole network. Finally, we will look at cases where multiple paths between pairs of nodes are considered in constructing DAG centrality measures.

3.1.1 Basic DAG centrality measures

- In- or out-degree. For example the citation index in citation networks.
- Use of transitive reduction or transitive completion.
- Volume measures, number of ancestors and descendants.

One of the most useful centrality measures is degree, just as it is with all networks. For a directed network such as a DAG, we can use both in-degree and of each node of (2.2.2) as centrality measures. Formally, for an unweighted DAG we have that

$$k_v^{(\text{in})} = \sum_u A_{vu}, \quad k_v^{(\text{out})} = \sum_u A_{uv}. \quad (3.1.1)$$

For instance, for a citation network defined as in section 2.1, the out-degree is the size of the bibliography of each document (node). For many documents, we might expect an authoritative source to have many references in its bibliography so we can see how out-degree might be a good measure of how useful a document is as a source of information, and thus, a good measure of how important each document might be in this information source sense. Equally, the number of times a document is cited by others, the in-degree, is often used as a measure of the impact of that document. For a food web the in-degree and out-degree tell us how many predator species and how many prey species each species has. So a top predator has zero out-degree given our definition of a food web given in section 2.1.

The in- or out-degree of a node is a local measure. That is it does not probe the whole DAG network and with it the unique sense of order in a DAG. So it is interesting to look for measures that probe the whole network structure.

We have already seen how a DAG has two special transformations, transitive completion and transitive reduction of section 2.7. In these processes, edges are removed or added depending on the existence of paths in the network and these paths do probe beyond nearest neighbours. So we can also investigate the degree of a node v after transitive reduction which we will call the REDUCED DEGREE , the number of essential edges at a node, so we have the reduced in-degree $k_v^{(\text{tr,in})}$ and reduced out-degree $k_v^{(\text{tr,out})}$. These have been used to give new insights into the impact of documents in a citation network (Clough et al., 2014). Formally, for an unweighted DAG we have that

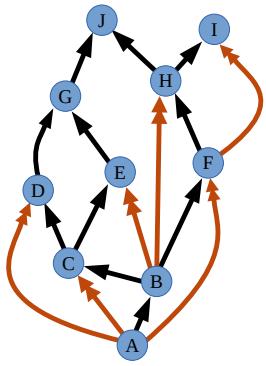
$$k_v^{(\text{tr,in})} = \sum_u A_{vu}^{(\text{tr})}, \quad k_v^{(\text{tr,out})} = \sum_u A_{uv}^{(\text{tr})}. \quad (3.1.2)$$

In principle we could also define the ENHANCED DEGREE of a node v as the degree after transitive completion. However, transitive completion means a node v is connected by an edge to every descendant node and has an edge ending at v from every ancestor node. That is enhanced in-degree $k_v^{(\text{tc,in})}$ is simply the number of ancestors $\tilde{N}_v^- = |\tilde{\mathcal{V}}^-(v)|$ while the enhanced out-degree $k_v^{(\text{tc,out})}$ is just the number of descendants, $\tilde{N}_v^+ = |\tilde{\mathcal{V}}^+(v)|$. We could even imagine another variant where we just count the number of source nodes $\mathcal{S} = \{v | k_v^{(\text{in})} = 0\}$ or sink nodes $\mathcal{T} = \{v | k_v^{(\text{out})} = 0\}$ in the ancestors or descendants, say $\hat{N}_v^- = |\tilde{\mathcal{V}}^-(v) \cap \mathcal{S}|$ and $\hat{N}_v^+ = |\tilde{\mathcal{V}}^+(v) \cap \mathcal{T}|$. Writing this out formally we have

$$\tilde{N}_v^- = |\tilde{\mathcal{V}}^-(v)| = \sum_{u|u \prec v} 1, \quad \tilde{N}_v^+ = |\tilde{\mathcal{V}}^+(v)| = \sum_{u|u \succ v} 1, \quad (3.1.3)$$

$$\hat{N}_v^- = |\tilde{\mathcal{V}}^-(v) \cap \mathcal{S}| = \sum_{u \in \mathcal{S} | u \prec v} 1, \quad \hat{N}_v^+ = |\tilde{\mathcal{V}}^+(v) \cap \mathcal{T}| = \sum_{u \in \mathcal{T} | u \succ v} 1. \quad (3.1.4)$$

However, these simple DAG centrality measures immediately show us a feature common to most DAG centrality measures. That is the order inherent in the DAG does have a strong effect



Node	$k^{(\text{in})}$	$k^{(\text{out})}$	$k^{(\text{tr,in})}$	$k^{(\text{tr,out})}$	\tilde{N}^-	\tilde{N}^+	$(\tilde{N}^-. \tilde{N}^+)$	\hat{N}_v^-	\hat{N}_v^+
A	0	4	0	1	0	9	0	0	2
B	1	4	1	2	1	8	8	1	2
C	2	2	1	2	2	4	8	1	1
D	2	1	1	1	3	2	6	1	1
E	2	1	1	1	3	2	6	1	1
F	2	2	1	1	2	3	6	1	2
G	2	1	2	1	5	1	5	1	1
H	2	2	1	2	3	2	6	1	2
I	2	0	1	0	4	0	0	1	0
J	2	0	2	0	8	0	0	1	0

Table 3.1: A table of node centrality values for the ten-node DAG shown here. The double-arrow edges in edges are those removed under transitive reduction. The alphabetical order by label of the nodes is one of the topological orders for this DAG. We can see how for most columns the numbers generally decrease or generally increase. The $k_v^{(\text{in})}$ and $k_v^{(\text{out})}$ are the in- and out-degree of (3.1.1) for each node v while $k_v^{(\text{tr,in})}$ and $k_v^{(\text{tr,out})}$ are the reduced degree, the same but measured in the transitively-reduced network. The \tilde{N}_v^- and \tilde{N}_v^+ are the number of ancestors and descendants of v as defined in (3.1.3) while \hat{N}_v^- is the number of sources in the ancestors of node v while \hat{N}_v^+ is the number of sinks in the descendants of node v , see (3.1.4).

on these measures, even the local measure such as degree. Essentially the later a node is in a DAG, as measured by height, depth or on some total order, the more nodes it has in its past, the more opportunities a node has for incoming edges. The classic example is a document in a citation network where older papers gather many more citations than newer papers simply because they have had more time to gather citations. Conversely, if we look a node v that is late in some total order, such late nodes have fewer nodes that can come even later so we might expect both out-degree and number of descendants to be limited. So one of the most important features of DAG node centrality measures is that they are invariably strongly correlated with their position in their DAG, as measured by any total order or similar measure. Some will be increasing with order, some will be decreasing with order. We could also imagine that some will mix this behaviour. So if we took the product of the numbers of descendants and ancestors, $\tilde{N}_v^-. \tilde{N}_v^+$, this measure of importance for node v will most likely peak for nodes in the “centre” of the DAG. (T)Plot these measures for a larger network. (end T)

(T)Produce code to calculate all these basic statistics. Run them on some standard DAGs. Plot results here. (end T)

3.1.2 Optimal path centrality measures

(T)Should this be single path not optimal path? (end T) (T)This is where we look at main path analysis. Need to reword this as cut and paste from paper draft. (end T) (T)Should this summary of properties go in the coda? (end T)

So far we have looked at centrality measures which did not involve paths. Standard network analysis offers a wide range of centrality measures based on the paths between a set of nodes. In this section we look at those measures that use a single path between a set of node pairs. In general there are many paths between any two nodes so by choosing one type of path means we are singling out that path type of path as “optimal” in some sense. This gives us some obvious variations for each centrality measures, as we can still use the shortest paths which are invariably used for these measures on a standard network but here with a DAG we have the longest path as a variable option. In both cases, the whole DAG is probed by considering a

large number of start and end nodes for each path and so both approaches are based on the properties of a large number of paths through the DAG. So we will call these OPTIMAL PATH CENTRALITY MEASURES and leave those measures that use multiple paths between every node pair considered — MULTIPLE PATH CENTRALITY MEASURES — to the next section.

Another option we have for most of these measures is to change the starting and end points for the paths. So in general we will work with a generic set of initial nodes \mathcal{I} and a set of final nodes \mathcal{F} . Typical examples will be to choose all nodes for both sets or to use the source nodes as the initial nodes $\mathcal{I} = \mathcal{S}$ and the sink nodes for the final nodes $\mathcal{F} = \mathcal{T}$.

Finally, we will have some sort of measurement made on each path. We then have to decide how to combine them, simply add up the results, take the average or even an extremal value such as the largest one. Again, we will see different choices made in different cases.

Betweenness

We looked at betweenness Freeman (1977, 1980) in section 1.3 where we saw that this standard centrality measure is based on the use of the shortest-path between two nodes. The shortest path between two nodes is perfectly well defined for a DAG so there is no problem in using the length of the shortest path to define betweenness as normal but in the context of a DAGs we will denote this as $b_v^{(sp)}$. This will then give us a set $\mathcal{P}^{(\text{opt})}$ of optimal paths. However, we have also seen in section 2.4.1 that unlike most networks, DAGs have a well defined longest path.^(T)¹ So we can now define betweenness in terms of the length of the longest path between two nodes and we will denote this as $b_v^{(lp)}$.

The other change is betweenness in traditional networks analysis uses all nodes as both possible initial and final nodes. This choice of initial nodes \mathcal{I} and final nodes \mathcal{F} for the optimal paths is still possible for a DAG. Of course for many such pairs there will be no such paths, e.g. from a sink node to a source node, but the normalisations in the formula can take account of this. Equally, it may well be that for a given question, the researcher may want to restrict the initial and final node set in some way. For instance, we could choose a more natural starting point for paths, the source nodes \mathcal{S} , and a natural set for the end of the optimal paths, the sink nodes \mathcal{T} . We will leave this choice open. This gives us a generalised version of the betweenness given in (1.3.2) for traditional networks.

Definition 3.1.1— Betweenness b_v

The BETWEENNESS b_v of a node v is the number of optimal paths from a set $\mathcal{P}^{(\text{opt})}(i, f)$ from an initial node $i \in \mathcal{I}$ to any final node $f \in \mathcal{F}$ which pass through node v , weighted by the inverse of the number of such paths from i to f .

$$b(v, \mathcal{P}^{(\text{opt})}) = \frac{1}{Z_b} \sum_{P \in \mathcal{P}^{(\text{opt})}(i, f)} \sum_{u \in P | u \neq i, f} \frac{1}{|\mathcal{P}_{\text{opt}}(i, f)|} \delta_{u,v}, \quad (3.1.5)$$

Here $|\mathcal{P}_{\text{opt}}(i, f)|$ is the number of such optimal paths that run from a given initial node i to a given final node f .

Let us unpack this rather unwieldy formula in more detail. The set of optimal paths $\mathcal{P}^{(\text{opt})}$ are typically the longest paths or the shortest paths between nodes. Here we restrict this to those optimal $\mathcal{P}^{(\text{opt})}(i, f)$ running between one of the initial nodes i to one of the final nodes f .

¹(T)Why is LP preferred?

From each of these optimal paths, P , we check to see if the optimal path passes through our chosen node v , i.e. if a node u on the path \P is equal to v as indicated by the delta function at the end of (3.1.5). It is common to *exclude* the initial and final nodes which we have done here in the constraint on the second sum but this is a further choice². The $1/|\mathcal{P}_{\text{opt}}(i, f)|$ factor allows for there to be more than one optimal path between initial and final node with this factor ensuring that each initial-final node pair can contribute a factor of between 0 and 1 to any node v . For both shortest and longest paths, if path length is measured in terms of the number of edges, ties in this integer-valued length are common so there could be many longest or shortest paths between two points. So this normalisation factor is often included and is non-trivial. The overall normalisation Z_b is some suitable constant normalisation which is needed if we compare networks of different sizes and is chosen to suit the context. For instance, the normalisation Z_b could be the number of pairs i, f of initial nodes i and final nodes f that are connected so that betweenness values lie between zero and one.

The traditional betweenness measure is $b^{(\text{sp})}(v) = b(v, \mathcal{P}^{(\min)})$ where the optimal paths from i to f are the paths with the shortest network length over all distinct edge pairs, that is over $\{i, f \in \mathcal{V} \times \mathcal{V} | i \neq f\}$. For an undirected network of a single component, every node v has $(N-1)(N-2)/2$ such pairs i, f (neither equal to v) which could contribute to the betweenness of v , so $0 \leq b(v, \mathcal{P}_{\text{opt}}) \leq (N-1)(N-2)/(2Z_b)$ and the upper limit is independent of v . However, when we have a DAG, the only paths that contribute are initial nodes which are ancestors of v and final nodes which are predecessors of v . Now we see that for a DAG we have that

$$0 \leq b^{(\text{DAG})}(v, \mathcal{P}_{\text{opt}}) \leq \frac{\tilde{N}_v^- \tilde{N}_v^+}{2Z_b}. \quad (3.1.6)$$

where \tilde{N}_v^- and \tilde{N}_v^+ from (3.1.3) are the number of ancestors and descendants of v .

We do need to adjust these measures slightly because the connectivity of a DAG is much more limited. The standard network measures using a single optimal path, previously just the shortest-path, used the paths between all pairs of nodes to calculate betweenness. Now we see that for betweenness the only paths that can pass through a node v are ones from an ancestor node of v in the past of v to a descendant node in the future of v . The previous expressions still work (modified for longest path if required) it is just that the optimal path between many pairs of nodes will simply have no chance of passing through the node of interest, v . For a DAG we can rewrite the betweenness expressions formally in a rather cumbersome way as³

That is the formula for betweenness still works but betweenness on a DAG is now profoundly bounded and so influenced by the position of the node in the order of the DAG. It is a prime example of a centrality measure in a DAGs that looks as if betweenness will be strongly correlated with $\tilde{N}_v^- \tilde{N}_v^+$. One response might be to include this factor in the normalisation, that is for a DAG it makes sense to make Z_b dependent on the node v as $Z_b = \tilde{N}_v^- \tilde{N}_v^+$.

In fact, unlike a standard network, because of the order in a DAG there is another set of potential initial and final points for the paths included in betweenness, and that is the set of ancestors which are sources $\tilde{N}_v^- \cap \mathcal{S}$, and the set of sinks which are descendants $\tilde{N}_v^+ \cap \mathcal{T}$. We will not consider this idea here but it is used in other measures, for example in main path analysis.

The other special feature about a DAG is that we have a second, more natural, set of optimal paths, that provided by the longest paths by number of edges (network length) between

²Initial and final nodes on the path are sometimes included in the betweenness expression (3.1.5). In the same way allowing trivial paths where $i = f$. For fully connected networks where $\mathcal{I} = \mathcal{F} = \mathcal{V}$ these changes add constant factors to the betweenness that are simple functions of the total number of nodes N . In other cases, such as DAGs, these choices will have a more complicated effect which effects how we compare betweenness values of nodes in different networks.

³We exclude the first and last node in any path when counting the number of paths passing through a node. These can be included if preferred but it just adds a factor equal to the number of ancestors plus the number of descendants to each node.

two points. So it is probably more useful to work in terms of a longest-path betweenness $b^{(lp)}(v) = b(v, \mathcal{P}^{(\max)})$ where now $\mathcal{P}^{(\max)}$ is the set of longest paths from any node i to any node f .

Closeness and Status

Closeness centrality (Bavelas, 1950), which we discussed in section 1.3, when applied to DAGs faces some of the same issues as faced by betweenness. First the order in the DAG means we can split closeness into two measures, one based on optimal paths from a node v to all other nodes, PAST CLOSENESS c_v^- , or looks at paths ending at v , FUTURE CLOSENESS c_v^+ .

Definition 3.1.2— Closeness c

The PAST CLOSENESS c_v^- of a node v is the distance d_{vi} from each initial node $i \in \mathcal{I}$ to the node v . Similarly, the FUTURE CLOSENESS c_v^+ of a node v is the distance d_{fv} from the node v to any final node $f \in \mathcal{F}$.

$$c_v^- = \tilde{N}_v^- \left[\sum_{u \in \mathcal{V}_v^-} d_{vu} \right]^{-1}, \quad c_v^+ = \tilde{N}_v^+ \left[\sum_{u \in \mathcal{V}_v^+} d_{uv} \right]^{-1}. \quad (3.1.7a)$$

Here d_{vu} is the distance from node u to node v . Two obvious choices for the distance d are to use the network length of the shortest path $d_{vu}^{(sp)}$ giving $c_v^{-(sp)}$ and $c_v^{+(sp)}$ or we use the network length of longest path $d_{vu}^{(lp)}$ giving $c_v^{-(lp)}$ and $c_v^{+(lp)}$. The normalisation is just the number of ancestors, $\tilde{N}_v^- = |\tilde{\mathcal{V}}^-(v)|$ or the number of descendants, $\tilde{N}_v^+ = |\tilde{\mathcal{V}}^+(v)|$. In principle we could use any distance measure we like, see section 3.3.2 for some more examples.

Interestingly, there is another view that leads to measures that are simply proportional to the inverse of closeness and these are the STATUS σ_v^- and CONTRASTATUS σ_v^+ measures defined by Harary (1959) (see also p204-205 in Wasserman and Faust (1994)) to analyse hierarchies in organisations. One of the ways Harary defined the status σ_v^- for a node v is that it is the sum of the distances from node u to all subordinates (down the organisational structure). Contrastatus σ_v^+ for a node v was for the opposite direction, the sums of distances to all superiors. Harary imagined his organisations as directed trees and we will define our edges as directed from immediate subordinate to immediate superior⁴. Then in our notation the definitions of status σ_v^- and contrastatus σ_v^+ become⁵

$$\sigma_v^- = \sum_{u|u \prec v} d(u, v) = \frac{\tilde{N}_v^-}{c_v^-}, \quad \sigma_v^+ = \sum_{u|u \succ v} d(v, u) = \frac{\tilde{N}_v^+}{c_v^+} \quad (3.1.8)$$

So status σ_v^- is the sum of the distances from ancestors in $\mathcal{V}^-(v)$ while contrastatus σ_v^+ is the sum of the distances to descendants in $\mathcal{V}^+(v)$. Each is simply the inverse of the corresponding closeness measure scaled by the total number of ancestors or descendants, which themselves are another non-path based centrality measure discussed in section 3.1.1.

It is important to note that Harary was only considering directed trees, the norm for organisational structures considered in Harary (1959). In that case there is either no path

⁴So if we were to follow our nomenclature, status is what we would call “past status” and contrastatus is our “future status” but we will stick the terms used in the literature.

⁵Harary was working in the context of a directed tree where between any two nodes there is either no path in either direction or there is just one path in one direction. Some of the other formulae in Harary (1959) are for directed trees only.

or just only one directed path between points. For the distance, Harary used the network length of the path (number of edges) between two nodes and in a tree this path is unique so $d(u, v) = d^{(sp)}(u, v) = d^{(lp)}(u, v)$. However, in most contexts, a DAG is not a tree and these distances are not the same. As we mentioned for closeness and discussed in section 3.3.2, there is a much wider range of distances we can use in a DAG.

(T)⁶

We can also see that there are some simple bounds on closeness and status. Limiting ourselves to network distance, number of edges in either longest or shortest paths, we see that the distance between any two nodes must be less than the height of the DAG, H . Therefore we see that

$$0 \leq \sigma_v^- \leq H\tilde{N}_v^-, \quad 0 \leq \sigma_v^+ \leq H\tilde{N}_v^+, \quad (3.1.9)$$

$$\infty \geq c_v^- \geq \frac{1}{H}, \quad \infty \geq c_v^+ \geq \frac{1}{H}. \quad (3.1.10)$$

While this is a very loose bound, we clearly see that status and contrastatus are likely to be correlated with the number of ancestors \tilde{N}_v^- or the number of descendants \tilde{N}_v^+ respectively. If we normalise the status and contrastatus by these factors, i.e work with the average distance to every reachable node (something called FARNESS), we are likely to remove this dependence and closeness contains that at the expense of having unbounded values (in principle). There is in any case good reasons to suppose that these measures are also correlated with in- or out-degree in many larger networks (Evans and Chen, 2022).

Closeness, or status and contrastatus, are example of centrality measures capturing something about the importance of a node in the network. Just as we did with the number of ancestors and the number of descendants, if we want a single overall measure of node importance, centrality, we can combine the two indices in various ways. For instance, working with status we might consider the sum ($\sigma_v^{-(lp)} + \sigma_v^{+(lp)}$), the difference (as used by Harary, his net status)(T)⁷ ($\sigma_v^{-(lp)} - \sigma_v^{+(lp)}$), the product ($\sigma_v^{-(lp)} \times \sigma_v^{+(lp)}$) and so forth. The product is particularly interesting as it contains contributions from every optimal path from every ancestor i to every descendant f provided this path visits node v . This is similar to but different from betweenness which also looks at optimal paths from every ancestor i to every descendant f but only when they are not constrained to pass through v .(T)⁸

Generalised height and depth

We have already seen in section 2.5 that we can assign a height and depth value to every node. The height $h(v)$ is the largest network length of any path from any source to a node v while the depth $d(v)$ is the largest network length of any path from a node to a sink. These are not generalisation of properties that we have in a standard network.

Height and depth fit into this section as they are based on the properties of a single optimal path from the node of interest v to some node f in a set of final nodes \mathcal{F} , or from an initial node i from some set of initial nodes \mathcal{I} to the node v of interest. As with most node measures

⁶(T)These status indices can easily be generalised to other contexts as as food webs (Harary, 1961) which we will discuss in section 5.3. In particular they will work for more complicated structures than the original trees of Harary (1959) and that includes more general DAGs. In most cases there is now more than one path between two nodes and now we can measure the distance in different way. Harary specifically suggests using the shortest path $d^{(sp)}(u, v)$ as the distance measure in Harary (1959) though as noted there is only one path between points so this is also the longest path. For his follow up work on food webs, (Harary, 1961), there can be different paths and so there is more than one way to generalise Harary's original status definitions.

⁷(T)Eqn 14 of (Harary, 1959).

⁸(T)We could look at the triangle identity deficit $d(f, i) - d(f, v) - d(v, i)$ which measure how far node v lies off the optimal path from i to f .

for a DAG, they give one measure, height $h(v)$ for each node v using information from the past of v while the complementary measure, here depth $d(v)$, is based only the future of v .

(T)??? Must the distance measure be a maximum over some path length ??? (**end T**)

So we could write generalised forms of height as $\psi_v^{-,\ell}$ and a generalised depth $\psi_v^{+,\ell}$ in terms of any given path weight $\ell(P)$ for some path P as follows.

Definition 3.1.3— Generalised height and depth I

The generalised height $\psi_v^{-,\ell}(\mathcal{I})$ of a node v is the largest path weight ℓ for any path from any initial node $i \in \mathcal{I}$ to the node v . Similarly, the generalised depth $\psi_v^{-,\ell}(\mathcal{F})$ of a node v is the largest weight of paths from the node v to any final node $f \in \mathcal{F}$.

$$\psi_v^{-,\ell}(\mathcal{I}) = \max_{P \in \mathcal{P}(v,i), i \in \mathcal{I}} (\ell(P)) \quad (3.1.11a)$$

$$\psi_v^{+,\ell}(\mathcal{F}) = \max_{P \in \mathcal{P}(f,v), f \in \mathcal{F}} (\ell(P)) \quad (3.1.11b)$$

Here $\mathcal{P}(v,i)$ is the set of all paths from all initial nodes $i \in \mathcal{I}$ to v , $\mathcal{P}(f,v)$ is the set of all paths from v to all final nodes $f \in \mathcal{F}$, and $\ell(P)$ is the path weight for path P .

Note in this definition we need to include the trivial paths (v) of network length zero and assume these are assigned zero weight. Otherwise source and sink nodes will not have zero generalised height and depth respectively. For the standard height $h(v)$ of (2.5.1) and depth $d(v)$ of (2.5.2) we just use the network length $\ell^{(\text{net})}(P)$ of a path P , the initial node set is just the global sources \mathcal{S} and the final node set are the sink nodes \mathcal{T} — that is $h(v) = \psi_v^{-,\max}(\mathcal{S})$ and $d(v) = \psi_v^{+,\max}(\mathcal{T})$.

As we will see with many (but not all) of the path based measures, there is also an alternative iterative definition that gives us a different view of height/dept and a useful starting point for numerical computations.

Definition 3.1.4— Generalised height and depth II

The generalised height $\psi_v^{-,\ell}(\mathcal{I})$ of a node v is the largest value of the height of parent neighbour node u plus the weight of the edge from u to v . In the same way the generalised depth $\psi_v^{+,\ell}(\mathcal{F})$ of a node v is the largest value of the depth of child neighbour node u plus the weight of the edge from v to u .

$$\psi_v^{-,\ell}(\mathcal{I}) = \begin{cases} \max_{u \in \mathcal{N}^-} (A_{vu} + \psi_u^{-,\ell}(\mathcal{I})) & \text{if } k_v^{(\text{in})} > 0 \\ 0 & \text{if } k_v^{(\text{in})} = 0 \end{cases}, \quad (3.1.12a)$$

$$\psi_v^{+,\ell}(\mathcal{F}) = \begin{cases} \max_{u \in \mathcal{N}^+} (A_{uv} + \psi_u^{+,\ell}(\mathcal{F})) & \text{if } k_v^{(\text{in})} > 0 \\ 0 & \text{if } k_v^{(\text{out})} = 0 \end{cases}. \quad (3.1.12b)$$

To show these definition 3.1.3 follows from 3.1.4 are equivalent we need to remember that the weight of a path is the sum of the weights of the edges in a path. Then, in a DAG all paths to a node v must pass through one of its parent nodes u . In addition, that edge from

parent node u to node v , must be the last edge in any path to v and it must be the one and only extra edge we have when comparing the paths to v and the paths to those neighbours u . Together these tell us that if we know the largest path weight of paths ending at neighbour u then (3.1.12a) follows for v when looking back to initial nodes in the past of v . Similar arguments work for (3.1.12b).

In particular, we may write the standard height $h(v)$ and depth $d(v)$ in a simple iterative form, that is

$$h(v) = \begin{cases} 1 + \max_{u|A_{vu}=1}(h(u)) & \text{if } k_v^{(\text{in})} > 0, \\ 1 & \text{if } k_v^{(\text{in})} = 0 \text{ and } v \in \mathcal{I}, \\ 0 & \text{if } k_v^{(\text{in})} = 0 \text{ and } v \notin \mathcal{I}, \end{cases} \quad (3.1.13)$$

$$d(v) = \begin{cases} 1 + \max_{u|A_{uv}=1}(d(u)) & \text{if } k_v^{(\text{out})} > 0 \\ 1 & \text{if } k_v^{(\text{out})} = 0 \text{ and } v \in \mathcal{F}, \\ 0 & \text{if } k_v^{(\text{out})} = 0 \text{ and } v \notin \mathcal{F}, \end{cases} \quad (3.1.14)$$

One special case is where the initial or final set are just a single node, as we have when we analyse an interval DAG. In that case the height and depth represent a type of distance measure between two nodes and it is better to think of the measures in terms of distances rather than height and depth.

Another variation is that we can replace the maxima by minima to produce a type of shortest path height and depth.

Coda: deconstructing optimal path centrality measures

(T)Sketch of options: past and future cones, all nodes vs source/sinks etc. **(end T)**

When we look at the measures in this section, we can see there are key choices made in all of these measures that suggest there are many possible variations.

The common feature of all the methods in this section is that in trying to define some generic measure, say m_v , for a node v they use a set of paths starting from some node set \mathcal{I} and a set of nodes for the end of paths \mathcal{F} . In each case they only use a single path between two nodes when in general there can be a large number of paths between any two nodes.

In this section we describe these methods as being optimal path centrality measures as we are singling out one type of path, so in some sense this is considered “optimal”. We could use the path used in the equivalent centrality measures used on standard networks, i.e. the shortest path. However, for DAGs we have another option which is usually more natural to use in this context, the longest path. So the choosing type of path to be used is the first option.

The next option we have is the choice of the set of initial nodes \mathcal{I} and the set of final nodes \mathcal{F} to be used. The obvious choices are either to use the set of all nodes in both cases $\mathcal{I} = \mathcal{F} = \mathcal{V}$ or to use the source nodes as the initial nodes $\mathcal{I} = \mathcal{S}$ and the sink nodes for the final nodes $\mathcal{F} = \mathcal{T}$.

Another option we have for the paths used is whether we look at paths from an initial node i to some final node f , as in betweenness, or whether we look at two paths, one from i to v and another from v to f as used in closeness, which leads to a pair of measures m_v^\pm based only on paths in the past or the future of v .

If our initial and final node sets have many nodes, then we will have a large number of paths and we will make some measurement on each path. We then need to combine the measurements made on each path to get a single result for our centrality measure. Some of the options we have seen are to simply add up the results, to take the average, or to take an extremal value such as the largest one, e.g as used for the height/depth/criticality measures.

A last point to note is that we could also think about finding these types of centrality measure on either the transitively reduced or transitively complete DAG, just as we did with degree. However, one of the key features is that the longest paths do not change under these transformations so these centrality measures will also be unchanged if we use the more natural longest path version of these measures. Shortest paths are not invariant under these transitive reduction and completion on a DAG so centrality measures built on such paths will change.

We have given some standard centrality measures, adapted for DAGs. We can see that in each case, the choices outlined above made for each measure are different. So there are many other variants. Whether other centrality measures in this wider family of optimal path centrality measures are useful will depend on the context and we will leave the topic here. Instead we will now move on to look at measures that use multiple paths between every node pair used to probe a DAG to produce a centrality measure.

3.1.3 Multiple Path Methods

In the previous section, we looked at ways to use a single optimal path between a pair of nodes in order to probe the global network structure to give a node a measure of its position, its centrality, in a network — optimal path centrality measures. In this section we will look at a variety of methods to do that but where we use *all* possible paths between each pair of nodes to do a similar type of analysis — MULTIPLE PATH CENTRALITY MEASURES.

Broadcast Path Count

To produce multiple paths between nodes, we will first use a broadcast process as described in section 1.4.1. This is easily expressed in terms of the adjacency matrix \mathbf{A} since for any unweighted network $[\mathbf{A}^n]_{vu}$ gives the number of paths from node u to node v which are of network length n . This matrix is the basis of eigenvalue centrality and Katz centrality for standard networks. For a DAG, this leads to two natural measures: we can ask how many distinct paths W_v^- arrive at a node v from the past and correspondingly, and how many paths leave node v , W_v^+ connecting it to the future. We will see later in section ?? that one of the uses for these node values W_v^\pm is a well known method to find the most “important” path in a network, a method known as main path analysis. (T)???

This gives us an interpretation of this count as a type of broadcast process. (end T)

Definition 3.1.5— Broadcast Path Count I

The past BROADCAST PATH COUNT centrality measure $\mathbf{W}_v^-(\mathbf{I})$ for a node v is defined to be the sum of the past broadcast count measures all parent nodes plus the v -th entry, I_v , of some specified initial vector \mathbf{I} . The future broadcast path count centrality measure $\mathbf{W}_v^+(\mathbf{F})$ of node v is the sum of the future broadcast count measures of all child nodes plus F_v .

Formally

$$W_v^-(\mathbf{I}) = I_v + \sum_u A_{vu} W_u^-(\mathbf{I}), \quad \mathbf{W}^- = \mathbf{I} + \mathbf{A} \cdot \mathbf{W}^-, \quad (3.1.15a)$$

$$W_v^+(\mathbf{F}) = F_v + \sum_u W_u^+(\mathbf{F}) A_{uv}, \quad (\mathbf{W}^+)^T = (\mathbf{I})^T + (\mathbf{W}^+)^T \cdot \mathbf{A}. \quad (3.1.15b)$$

For simplicity we will sometimes drop the explicit dependence on the initial vector \mathbf{I} or final

vector \mathbf{F} , e.g. $\mathbf{W}^- \equiv \mathbf{W}^-(\mathbf{I})$.

We can interpret these expressions (3.1.15) in terms of the number of paths to and from a given node v . This leads us to an alternative definition.

Definition 3.1.6— Broadcast Path Count II

The past BROADCAST PATH COUNT centrality measure $\mathbf{W}_v^-(\mathbf{I})$ for a node v is defined to be the number of paths to v from every vertex u weighted by I_u summed over all nodes u . The future broadcast path count centrality measure $\mathbf{W}_v^+(\mathbf{F})$ for node v is the sum of the number of paths to every vertex u weighted by F_u and then summed over all nodes u .

Formally

$$W_v^-(\mathbf{I}) = \sum_u \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right]_{vu} I_u, \quad \mathbf{W}^- = \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right] \cdot \mathbf{I}, \quad (3.1.16a)$$

$$W_v^+(\mathbf{F}) = \sum_u F_u \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right]_{uv}, \quad (\mathbf{W}^+)^T = \mathbf{F}^T \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right]. \quad (3.1.16b)$$

To see that these two definitions are the same, we start from (3.1.16a) for \mathbf{W}^- and since $\mathbf{A}^0 = \mathbf{1}$ we see that

$$W_v^- = I_v + \sum_w \left[\sum_{n=1}^{\infty} \mathbf{A}^n \right]_{vw} I_w = I_v + \sum_u \sum_w A_{vu} \left[\sum_{n=0}^{\infty} \mathbf{A}^n \right]_{uw} I_w \quad (3.1.17)$$

$$\Rightarrow W_v^- = I_v + \sum_u A_{vu} W_u^-. \quad (3.1.18)$$

This is (3.1.15a). The equivalent result follows in the same way for the future broadcast path count \mathbf{W}^+ and so the two definitions 3.1.5 and 3.1.6 are equivalent.

The next choice is to specify the initial and final vectors, usually by specifying a set of equally weighted initial \mathcal{I} and final \mathcal{F} node sets. For a DAG there are two natural sets. First the set of all nodes \mathcal{V} can be used for both initial and final nodes, though in practice only the ancestors and descendants of v will contribute to these path count measures W_v^- and W_v^+ respectively. The other obvious set is to use the sources \mathcal{S} as the initial set \mathcal{I} and sinks \mathcal{T} as the set as final nodes \mathcal{F} . In either case we set $I_i = 1$ if node i is in the initial node set \mathcal{I} and $F_f = 1$ if node f is in the final node set \mathcal{F} , with other entries in these vectors zero. We might guess that the more nodes from the initial node set \mathcal{I} are in the past of a node v , $|\mathcal{I} \cap \tilde{\mathcal{V}}^-|$, then the higher the value of the past broadcast path count W_v^- . Similarly, we expect larger W_v^+ values for larger $|\mathcal{F} \cap \tilde{\mathcal{V}}^+|$. This hints that these centrality measures often show (non-linear) correlations with the simpler centrality measures such as \tilde{N}^\pm and \hat{N}^\pm of (3.1.3) and (3.1.4).

To give this broadcast path count measure some context, and to illustrate one type of process on a DAG, we imagine a process where each node v at time t has a value $w_v^-(t)$. At the next time step, we copy this value to each of the children of node v and they add all incoming values to give their new value at the next time step ($t + 1$). Then we have that

$$w_v^-(t + 1) = \sum_u A_{vu} w_u^-(t). \quad (3.1.19)$$

The $-$ superscript indicates that, because of the order in a DAG, the value $w_v^-(t)$ of node v at time t comes from values that have flowed from nodes in the *past* of v , from the ancestor nodes

in $\tilde{\mathcal{V}}^-$. By iterating this equation, and if we start with value $w_v^-(t=0) = I_v$ for node v at time t , then after t steps we have⁹

$$w_v^-(t) = \sum_u [\mathbf{A}^t]_{vu} I_u. \quad (3.1.20)$$

Note that \mathbf{A}^t is the matrix \mathbf{A} multiplied by itself t -times, while we denote the transpose of a matrix \mathbf{A} as \mathbf{A}^T . Remember that if a network is unweighted then $(\mathbf{A}^n)_{fi}$ is the number of paths from node i to node f . So if I_u is one only for some nodes in an initial node set \mathcal{I} , then $w_v^-(t)$ is the number of paths of network length t from any initial node i to v .

We can do the same but for a process where values flow in the reverse direction, against the direction of the edges in a DAG. Then for this process we denote the value each node v at time t as $w_v^+(t)$, since this depends on the values flowing from nodes in the *future* of v . This then gives us that

$$w_v^+(t+1) = \sum_u w_u^+(t) A_{uv} \quad (3.1.21)$$

where we note that the indices on the adjacency matrix \mathbf{A} in (3.1.19) are reversed compared to the forward flowing process (3.1.19). Now we need to specify the node values at time $t = 0$ using a vector \mathbf{F} and then iteration of (3.1.21) gives

$$w_v^+(t) = \sum_u F_u(t) [(\mathbf{A})^t]_{uv} = \sum_u F_u(t) [(\mathbf{A})^t]_{uv}. \quad (3.1.22)$$

An alternative interpretation is that the first form in (3.1.20), the value $w_v^-(t)$ is counting the paths of network length t that end at v but start from a node in some initial node set \mathcal{I} provided $I_i = 1$ if $i \in \mathcal{I}$ but is otherwise zero. In the same way (??) shows that the value $w_v^+(t)$ is counting the paths of network length t that start at node v but end at a node f in some final node set \mathcal{F} where $F_f = 1$ for $f \in \mathcal{F}$ and is zero for nodes not in \mathcal{F} .

The past broadcast path count centrality measure then comes from the total value $W_v^-(\mathbf{I})_v$ flowing though node v from nodes in the past of v , flowing from initial values encoded in a vector \mathbf{I} . Correspondingly, the total value $W_v^+(\mathbf{F})_v$ flowing backwards to v from nodes in the future of v with initial values \mathbf{F} . It follows from (3.1.20), and the equivalent for $w_v^+(t)$, that

$$W_v^-(\mathbf{I}) = \sum_{t=0}^{\infty} w_v^-(t) = \sum_u \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right]_{vu} I_u, \quad \mathbf{W}^-(\mathbf{I}) = \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right] \cdot \mathbf{I}, \quad (3.1.23a)$$

$$W_v^+(\mathbf{F}) = \sum_{t=0}^{\infty} w_v^+(t) = \sum_u F_u \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right]_{uv}, \quad (\mathbf{W}^+)^T = \mathbf{F}^T \cdot \left[\sum_{t=0}^{\infty} (\mathbf{A})^t \right]. \quad (3.1.23b)$$

So through the definition 3.1.6 of the broadcast path count centrality measures can be thought of as based on a broadcast process on the DAG.

(T)¹⁰

It is worth noting that though \mathbf{A}^n plays a role in the broadcast path count centrality measures and standard centrality measures such as eigenvalue centrality, they differ in the the former looks at paths of finite lengths while the latter focusses on long paths $t \rightarrow \infty$. This is because $\mathbf{A}^n = 0$ for $n > H$ of a DAG of height H so traditional eigenvector centrality cannot be used

⁹In principle the notation $w_v^-(t, \mathbf{I})$ would be better to indicate explicitly that the value depends on the initial values used. For simplicity, we drop this explicit reminder from the notation.

¹⁰(T)In some sense the broadcast path count measures are a generalisation from counting nodes, as in the number of ancestors and descendants of each node, \tilde{N}_v^\pm in (3.1.3)(T)¹¹ to counting paths from nodes in the past or future. Put another way, more nodes there are in the past of a node v , the larger we might expect W_v^- to be, W_v^+ is likely to be higher for node with a large number of descendants. So we should expect to see some correlation between W_v^- and \tilde{N}^- , likewise between W_v^+ and \tilde{N}^+ .

for a DAG. The long time limit of the broadcast process we use here, forwards or backwards has $w_v^\pm(t) = 0$ for $t > H$. That is the values run off the start or end of a DAG. DAGs have an order which implies they have a beginning (sources) and an end (sinks) and standard networks do not have this feature.

Many more variations are possible. As mentioned before we can look at other combination of our two measures for each node, say $W_v^- \pm W_v^+$ or $W_v^- \times W_v^+$. If our DAG is weighted, then the formulae still work but now every path has a length or weight equal to the products of the edge weights along that path¹². Equally, although we envisaged the initial and final nodes to be represented by 1 or 0 entries in the vectors \mathbf{I} and \mathbf{F} , the formulae work for any weighting of these initial and final nodes.

Perhaps the most common variation is to change matrix used in these expressions. The simplest example would be to weight paths of length t by a factor α^t for some fixed parameter α which means we replace \mathbf{A} by $\alpha\mathbf{A}$ in all the expressions for broadcast path count above. This is the technique used to define Katz centrality on standard networks. However, for a DAG we can choose α to have any value large or small since α^H is the largest factor we will need and this is finite. With a standard networks, we take $t \rightarrow \infty$ to find Katz centrality measures so $\alpha < 1$ is required.¹³

There are other ways to weight paths differently, and hence replace \mathbf{A} in these expressions. These are usually best considered as changing the process from the broadcasting process considered here. So we now move on to consider an example using a different process.

Diffusion path counts and Keystone centrality

(T) Do we use in/out or \pm for \mathbf{T} labels? NO But we do use Left/right multiplication not transpose. (end T)

(T) Do we refer to node-controlled diffusion rather than diffusion? (end T)

Let us move away from the broadcast process of Fig. 1.12 encoded by the adjacency matrix \mathbf{A} , to one using a diffusion process such as the one shown in Fig. 1.13. For a directed graph like a DAG, diffusion is described by two TRANSFER MATRICES. One is the forward transfer matrix $\mathbf{T}^{(\text{out})}$ defined in (1.4.3) which encodes diffusion in the direction of the edges as described in section 1.4.3. The other is the backwards transfer matrix $\mathbf{T}^{(\text{in})}$ corresponds to diffusion in the opposite direction to the edges. For an unweighted their definition in our notation is

$$\mathbf{T}_{vu}^{(\text{out})} = \begin{cases} \frac{1}{k_u^{(\text{out})}} A_{vu} & \text{if } k_u^{(\text{out})} > 0 \\ 0 & \text{if } k_u^{(\text{out})} = 0 \end{cases}, \quad (3.1.24a)$$

$$\mathbf{T}_{vu}^{(\text{in})} = \begin{cases} \frac{1}{k_v^{(\text{in})}} A_{vu} & \text{if } k_v^{(\text{in})} > 0 \\ 0 & \text{if } k_v^{(\text{in})} = 0 \end{cases}. \quad (3.1.24b)$$

Our definitions mean that $\mathbf{T}_{vu}^{(\text{out})}$ and $\mathbf{T}_{vu}^{(\text{in})}$ are non-zero only if there is an edge from node u to node v . Should we have a weighted network, we just replace degree by strength as shown in (1.4.3) and written out in (3.1.64). These transfer matrices are useful in many other situations e.g. the transfer matrix $\mathbf{T}^{(\text{out})}$ lies at the heart of the standard PageRank centrality measure.

What we are aiming for here is for DIFFUSION PATH COUNT centrality measure. One, X_v^- , is for diffusion of values going forwards but which count up values flowing towards v from the past

¹²This is not always what is required but if the edge weights are exponentials of a value of interest, the path weight is then the exponential of the sums of values required. Using appropriate definitions of edge weights can help us reach more relevant results.

¹³To be more precise $|\alpha| < 1/\lambda_1$ is needed where λ_1 is the largest eigenvalue of \mathbf{A} . We assume that $\lambda_1 > 1$ and is real which follows if the adjacency matrix is a non-negative matrix and the strength of every node (in- and out-) is at least 1.0, which is true in almost every practical example.

(hence the $-$ subscript). The second X_v^+ will be for reverse diffusion, that is counting up values flowing from the future of v backwards along links towards v . The algebraic manipulations to find the relevant expressions are identical to those used for the broadcast path counts, we are just replacing \mathbf{A} by $T^{(\text{out})}$ in the forwards process, and by $T^{(\text{in})}$ in the reverse process. We will run through these quickly just to be explicit, highlighting the different interpretation for this diffusion case.

We will start by interpreting these diffusion path count measures \mathbf{X}^\pm in terms of the value $x_v^-(t)$ of one node v at time t being split up and shared equally between the child nodes. So the value on the network is neither increased or decreased. The exception is if a node is a sink and has no child nodes. In this case only (for forwards diffusion) the value is lost and the total value on the network decreases. We can also think of $x_v^-(t)$ as being the *average* number of random walkers at a site v after t steps, a view that can match some real world contexts nicely¹⁴. This gives us the process considered by Gualdi et al. (2011) based on this forwards diffusion process where

$$x_v^-(t+1) = \sum_u T_{vu}^{(\text{out})} x_u^-(t) \quad (3.1.25)$$

Again, the $-$ subscript notation reminds us that the values $x_v^-(t)$ come from nodes in the past of node v . Note how the definition of $T^{(\text{out})}$ means that value $x_u^-(t)$ of any node u at one time t , is always split up equally and given to child nodes of u for the next step in a forwards diffusion process. The exception is when the node u has no child nodes when the value is lost. So we have conservation of the total value flowing on the DAG until values reach a sink node.

For the backwards diffusion process we find that

$$x_v^+(t+1) = \sum_u x_u^+(t) T_{uv}^{(\text{in})} \quad (3.1.26)$$

where now $x_v^+(t)$ values come from nodes in the future of node v . Again, we have conservation of values in this backwards diffusion process until values are lost but now at a source node.

If we let the initial values for the forwards diffusion process be given by vector \mathbf{I} , and for the reverse process by vector \mathbf{F} at $t = 0$, then

$$x_v^-(t) = \sum_u [(\mathbf{T}^{(\text{out})})^t]_{vu} I_u, \quad x_v^+(t) = \sum_u F_u [(\mathbf{T}^{(\text{in})})^t]_{uv} \quad (3.1.27)$$

Again, $(\mathbf{T}^{(\text{out})})^t = (\mathbf{T}^{(\text{out})})^t = 0$ for $t > H$ so we can't look at the long-time limit as we can with standard networks as used in PageRank and similar centrality measures. The equilibrium limit is trivial as values fall off the end of the DAG in forward diffusion, off the start with backwards diffusion, and no value is left on the DAG after more than H steps. So instead we count the total value that passed through each node v in this process. That is

$$X_v^- = \sum_{t=0}^{\infty} x_v^-(t), \quad X_v^+ = \sum_{t=0}^{\infty} x_v^+(t). \quad (3.1.28)$$

This gives us the first definition of DIFFUSION PATH COUNT centrality measure.

¹⁴The diffusion we look at here is a type of Markov process, a simple random walk, and is a stochastic process. A single physical random walker cannot be split up into fractional pieces as we do here in some of our equations. So what we are doing is looking at many different experiments where a random walker can make different choices in each experiment. We look at what happens on average over a large (infinite) number of experiments so then these average number of walkers $x_v^-(t)$ can be fractional.

Definition 3.1.7— Diffusion Path Count

The past DIFFUSION PATH COUNT centrality measure X_v^- for node v is the total value that diffuses forward through node v from initial values described by \mathbf{I} . Likewise, the future diffusion path count centrality measure X_v^+ for node v is the total value that diffuses backwards through node v from initial values described by \mathbf{F} .

Formally

$$X_v^- = \sum_u \left[\sum_{t=0}^{\infty} (\mathbf{T}^{(\text{out})})^t \right]_{vu} I_u, \quad \mathbf{X}^- = \left(\sum_{t=0}^{\infty} (\mathbf{T}^{(\text{out})})^t \right) \cdot \mathbf{I}, \quad (3.1.29a)$$

$$X_v^+ = \sum_u F_u \left[\sum_{t=0}^{\infty} (\mathbf{T}^{(\text{in})})^t \right]_{uv}, \quad (\mathbf{X}^+)^T = \mathbf{F}^T \cdot \left(\sum_{t=0}^{\infty} (\mathbf{T}^{(\text{in})})^t \right). \quad (3.1.29b)$$

Using the same algebra as for the broadcast path count node measures W_v^\pm of (??), we can rewrite these expressions for the diffusion path counts \mathbf{X}^\pm in terms of an iterative expression, useful for practical calculations. (T)¹⁵ That is

$$\mathbf{X}^- = \mathbf{I} + \left(\sum_{t=1}^{\infty} (\mathbf{T}^{(\text{out})})^t \right) \cdot \mathbf{I} = \mathbf{I} + \mathbf{T}^{(\text{out})} \cdot \left(\sum_{t=0}^{\infty} (\mathbf{T}^{(\text{out})})^t \right) \cdot \mathbf{I} \quad (3.1.30)$$

$$\Rightarrow \mathbf{X}^- = \mathbf{I} + \mathbf{T}^{(\text{out})} \cdot \mathbf{X}^- \quad (3.1.31)$$

where $(\mathbf{T}^{(\text{out})})^0 = \mathbf{1}\mathbf{l}$. This leads to the following definition (a generalisation of Gualdi et al. (2011)).

Definition 3.1.8— Diffusion Path Count centrality measure

The past diffusion path count centrality measure X_v^- is the sum of the fractions of the parent node values plus a the initial vector contribution I_v . The future diffusion past count centrality measure X_v^+ is the sum of the fractions of the child node values plus the final vector contribution F_v .

Formally

$$X_v^- = I_v + \sum_u T_{vu}^{(\text{out})} X_u^-, \quad \mathbf{X}^- = \mathbf{I} + \mathbf{T}^{(\text{out})} \cdot \mathbf{X}^-, \quad (3.1.32a)$$

$$X_v^+ = F_v + \sum_u X_u^+ T_{uv}^{(\text{in})}, \quad \mathbf{X}^+ = \mathbf{F} + \mathbf{X}^+ \cdot \mathbf{T}^{(\text{in})}. \quad (3.1.32b)$$

Note that, in the manner of a forward diffusion process, the contribution X_u^- from any node u is divided in proportion to the edge weights leaving u . So an edge (u, v) with weight A_{vu} means that node v will pick up a fraction $A_{vu}/s_v^{(\text{out})} = T_{vu}^{(\text{out})}$ of the value X_u^- .

We can find then some simple limits on the values of the diffusion path counts provided we are not dealing with negative values. We will assume¹⁶ that our DAG has non-negative

¹⁵(T)A comment on the notation. The X_v^- is the measure for node v that depends on random walkers coming forward from ancestors of node v hence we use the $-$ superscript on this measure. In the same way X_v^+ is the counting random walkers passing v that are moving backwards and come from descendants of v .

¹⁶This is the case for most types of analysis. However, negative weights can be useful but this is the focus of another topicas the topic, that of SIGNED NETWORKS.

weights, so $A_{vy} \geq 0$, and the initial and final values are also non-negative, $I_v, F_v \geq 0$. Let us first focus on the forwards diffusion and on the value X_v^- . We have noted that the value of each node is always split up equally and given to child nodes at each step in a forwards diffusion process as described in (3.1.26). So, in general the value stored at a node is conserved in this diffusion process except at sink nodes where values are lost on the next time step. In terms of paths, the value I_i from any initial node i will be split up and shared amongst the different paths leaving node i . If a node v appears somewhere on each of these paths, then the whole of the contribution I_i from node i will pass through node v at some point and so we get a full I_i contribution to X_v^- of node v . As we have a DAG, there are no cycles, so each path can pass through v at most once. However, in most cases, only some of these paths will flow through any given node so in general each node v will get less than a full contribution to X_v^- coming from of I_i of each node i . Similar arguments apply to X_v^+ and \mathbf{F} .

$$0 \leq I_v \leq X_v^- \leq \sum_u I_u, R \quad 0 \leq F_v \leq X_v^+ \leq \sum_u F_u. \quad (3.1.33)$$

However, we can do better than this as only ancestor nodes from the past of v can contribute their \mathbf{I} values to X_v^- . Equally only descendant nodes in the future of v give their \mathbf{F} values to X_v^+ .

$$0 \leq X_v^- \leq \sum_{u|u \preceq v} I_u \leq \sum_u I_u, \quad 0 \leq X_v^+ \leq \sum_{u|u \succeq v} F_u \leq \sum_u F_u. \quad (3.1.34)$$

For instance, consider the simple case where we let the initial and final node set be all nodes, $\mathcal{I} = \mathcal{F} = \mathcal{V}$ where $I_i = 1$ for every node i in the initial set \mathcal{I} , $F_f = 1$ for every node f in the final node set \mathcal{F} , with zeros otherwise in \mathbf{I} and \mathbf{F} . Then the limits are simply that $1 \leq X_v^- \leq N_v^- = 1 + \tilde{N}^-$ and $1 \leq X_v^+ \leq N_v^+ = \tilde{N}^+ + 1$, the number of nodes in the past and future of each node. This includes v itself ($I_v, F_v = 1$ here) but is simply one more than the number of ancestors nodes \tilde{N}^- and the number of descendant nodes \tilde{N}^+ from (3.1.3) which were suggested as simple centrality measures in section 3.1.1. Now we really do expect a strong (possibly non-linear) correlation between these centrality measures, X_v^- linked with \tilde{N}^- and X_v^+ with \tilde{N}^+ .

(T)¹⁷

In fact there is one particular case of these diffusion path counts which gives a popular centrality measure used in food webs and ecology known as the KEYSTONE INDEX K_v for node v as defined by¹⁹ Jordan et al. (1999) (see also Harary (1961), Libralato et al. (2006), Vasas and Jordán (2006), Fedor and Vasas (2009)). We will discuss food webs later in section ??.

For a general DAG, the keystone index K_v for a node v is the sum of two parts,

$$K_v = K_v^{(+)} + K_v^{(-)}. \quad (3.1.35)$$

The idea in the ecological context is that the importance of a change in a prey species u should be spread in proportion to the strength of interaction A_{vu} with the predators v of species u . If we write our DAG with edges from prey to predators, and if the value I_u represents a small change in the total energy available in species u then $T_{vu}^{(\text{out})} I_u$ represents the effect of this

¹⁷(T)???To finish our aside on these diffusive matrices, $\mathbf{T}^{(\text{in})}$ represents the same process but on the REVERSE DAG where all edges are reversed. (T)¹⁸ That is it represents a process where some value on the nodes being divided up equally and shared with the parent nodes.

¹⁹This keystone measure was inspired by the status-contrastatus measures of Harary (1961) in his work to analyse hierarchies in organisations Harary (1959), see section 3.1.2. However, as we have noted, status uses a single optimal path between nodes while keystone uses many paths between node pairs, so there are important differences.

perturbation on a predator v of species u . However, that will have a knock on effect as predators of v will benefit indirectly through a second order effect so we should also consider $(\mathbf{T}^{(\text{out})})^2 \mathbf{I}$ and so on. These effects are simply to be added to give a keystone index based on past nodes of $K_v^{(-)}$ giving us

$$K_v^{(-)} = \sum_u \left[\sum_{n=1}^{\infty} (T^{(\text{out})})^n \right]_{vu} \mathbf{1}_u = \sum_u [T_{vu}^{(\text{out})} (K_u^{(-)} + 1)]. \quad (3.1.36)$$

Here $\mathbf{1}$ is the vector with all ones as entries so $\mathbf{1}_u = 1$. As before we can find a second definition and iterative form for the past Keystone index. We see that

$$\mathbf{T}^{(\text{out})} \cdot \mathbf{K}^{(-)} = \sum_{n=2}^{\infty} (\mathbf{T}^{(\text{out})})^n \cdot \mathbf{1} \quad (3.1.37)$$

so if we add $\mathbf{T}^{(\text{out})} \cdot \mathbf{1}$ to this we get

$$\mathbf{T}^{(\text{out})} \cdot \mathbf{K}^{(-)} + \mathbf{T}^{(\text{out})} \mathbf{1} = \sum_{n=1}^{\infty} (\mathbf{T}^{(\text{out})})^n \cdot \mathbf{1} = \mathbf{K}^{(-)}. \quad (3.1.38)$$

So

$$\mathbf{K}^{(-)} = \mathbf{T}^{(\text{out})} \cdot (\mathbf{K}^{(-)} + \mathbf{1}) \quad (3.1.39)$$

is just an alternative iterative form, and a second definition, of this past keystone index $\mathbf{K}^{(-)}$.

The same perturbation in species u will also have an effect on the populations of the prey of species v , say w where $A_{uw} > 0$. This should be a negative effect on the population of w . However, the aim of the keystone index is not to perform some careful stability analysis of the species populations represented in a food web. Rather, the keystone index K_u for some species u seeks to show if changes in this species u has the largest effect (positive or negative) on the foodweb as a whole. So this keystone index chooses to count the effect of a species u on its predators as positive. Thus w scores $F_u T_{uw}^{(\text{in})} \geq 0$ for a change F_w in one of its predators even though an increase in u should have a negative effect on the population of w . Again, we want to pick up indirect effects too so we count these through $F_u (T_{uw}^{(\text{in})})^t$ for all path lengths t . The last assumption is that we want to treat perturbations in all species equally and count their direct and indirect effects on other species equally. That is, we choose $I_v = F + v = 1$. So

$$\mathbf{K}^{(+)} = (\mathbf{T}^{(\text{in})})^T \cdot (\mathbf{K}^{(+)} + \mathbf{1}) = \left(\sum_{n=1}^{\infty} [(\mathbf{T}^{(\text{in})})^T]^n \right) \cdot \mathbf{1}. \quad (3.1.40)$$

So the relationship between the keystone indices K_v^{\pm} and the diffusion path counts X_v^{\pm} is simply that the keystone index is defined using all nodes as being in the both initial and final nodes, $I_i = 1$, $F_f = 1$ for all $i, f \in \mathcal{V}$. The only difference between our diffusion path counts \mathbf{X}^{\pm} and the keystone index is that we do not count the trivial path from v to itself in the keystone counts K_v^{\pm} whereas that is included in the diffusion path counts X_v^{\pm} . More precisely

$$K_v^- = X_v^- (\mathbf{I} = \mathbf{1}) - 1, \quad K_v^+ = X_v^+ (\mathbf{F} = \mathbf{1}) - 1. \quad (3.1.41)$$

The future and past keystone indices²⁰ \mathbf{K}^{\pm} are simply a particular case of diffusion paths counts. The actual keystone index is simply the sum of these two parts. Interestingly, taking the

²⁰The use of the terms bottom and top leads to the standard notation $K_v^{(+)} \equiv K_v^{(\text{b})}$ and $K_v^{(-)} \equiv K_v^{(\text{t})}$ used here but this is just one view of the DAG. So in other contexts this may be the opposite of how you view the network. The relationship between a node and its ancestor nodes $K_v^{(+)}$ as this value for node v depends on the descendants of v so it is at the ‘bottom’ of the nodes higher in the hierarchy. The second measure $K_v^{(-)}$ looks at the descendants of node v , and imagines node v is above subordinate nodes in some hierarchy.

sum of the past and future centrality measures is just one of many ways of combining past and future versions of DAG centrality measures to get a single index for each node. For instance we might try to argue that $K_v^{(+)} - K_v^{(-)}$ is a better measure of the net change in populations since a change in v will increase descendant node populations but will decrease ancestor nodes. However, again we stress that the keystone index is not a good way to assess the actual perturbations in populations in some dynamical model based on a food web. The keystone index has a much simpler goal of trying give an importance score, a rating, related to the scale of changes in the food web when one species is perturbed, not a precise measure of the actual perturbations on populations. The keystone index is meant to give a rough feeling for the information in a food web DAG independent of further modelling assumptions.

So now we can write out the diffusion path count expressions for this special case of the keystone measures.

$$K_v^- = \sum_u \left[\sum_{n=1}^{\infty} (\mathbf{T}^{(\text{out})})^n \right]_{vu}, \quad (3.1.42\text{a})$$

$$K_v^+ = \sum_u \left[\sum_{n=1}^{\infty} (\mathbf{T}^{(\text{in})})^n \right]_{vu}. \quad (3.1.42\text{b})$$

This interpretation immediately tells us that K_v^- is zero only for source nodes. Equally, if every ancestor node starts with value 1 the most that can come through to any node v is equal to the number of ancestor nodes. That is

$$0 \leq K_v^- \leq \tilde{N}^- = |\tilde{\mathcal{V}}^-| \quad K_v^- = 0 \Leftrightarrow v \in \mathcal{S}, \quad (3.1.43\text{a})$$

$$0 \leq K_v^+ \leq \tilde{N}^+ = |\tilde{\mathcal{V}}^+| \quad K_v^+ = 0 \Leftrightarrow v \in \mathcal{T}. \quad (3.1.43\text{b})$$

$$K_v^{(-)} = \sum_u T_{vu}^{(\text{out})}(1 + K_u^{(-)}), \quad \mathbf{K}^{(-)} = \mathbf{T}^{(\text{out})} \cdot (\mathbf{K}^{(-)} + 1), \quad (3.1.44\text{a})$$

$$K_v^{(+)} = \sum_u (1 + K_u^{(+)})T_{uv}^{(\text{in})}, \quad \mathbf{K}^{(+)} = (\mathbf{K}^{(+)} + 1) \cdot \mathbf{T}^{(\text{in})}. \quad (3.1.44\text{b})$$

The same interpretation but in reverse, for flows in the opposite direction to the arrows in the DAG, apply for the K_v^+ keystone index so we have simply quoted the equivalent results here (\mathcal{T} is the set of sink nodes).

Averaging Neighbour Values

We have seen that we can build the diffusion path counts X_v^\pm of (3.1.32) out of the stochastic matrices $\mathbf{T}^{(\text{out})}$ and $\mathbf{T}^{(\text{in})}$ of (1.4.3). This gives us one measure $X_v^-(\mathbf{I})$ which is the average number of random walkers arriving when they move forward from initial nodes in the past of v while $X_v^+(\mathbf{F})$ counts the average number of random walkers arriving when they move backwards from nodes in the past of v . These diffusion path counts X_v^\pm of (3.1.32) look at the average number of random walkers *arriving* at v . This suggests there are two more options for node measures that we can form from the stochastic matrices $\mathbf{T}^{(\text{out})}$ and $\mathbf{T}^{(\text{in})}$. While we can phrase this in terms of random walks²¹ it is much simpler to phrase this as defining the new node measure $Y_v^-(\mathbf{I})$ for node v to be the *average* of all the values $Y_p^-(\mathbf{I})$ held by parent nodes p plus

²¹We ask what is the probability that random walker *leaving* a node v and moving forward will arrive at a final node \mathcal{F} , captured by $Y_v^+(\mathcal{F})$ or we look at how often random walkers leaving v and moving backwards arrive at initial nodes, something we will encode as $Y_v^-(\mathbf{I})$.

a given value I_v used to initialise this iterative definition. Likewise, $Y_v^+(\mathbf{F})$ for node v is the average of all the child node values $Y_e^+(\mathbf{F})$ plus a given value F_v .

Definition 3.1.9— Average neighbour values Y_v^\pm

The AVERAGE NEIGHBOUR VALUE centrality measure for the past, $Y_v^-(\mathbf{I})$ for node v , is given by the average of the values of the parent nodes plus the contribution I_v from the initial vector for node v . Similarly, the centrality measure $Y_v^+(\mathbf{F})$ for node v is given by the average of the values of the child nodes plus the contribution F_v from the final vector for node v .

$$Y_v^-(\mathbf{I}) = I_v + \sum_u T_{vu}^{(\text{in})} Y_u^-(\mathbf{I}), \quad \mathbf{Y}^- = \mathbf{I} + \mathbf{T}^{(\text{in})} \cdot \mathbf{Y}^-, \quad (3.1.45a)$$

$$Y_v^+(\mathbf{F}) = F_v + \sum_u Y_u^+(\mathbf{F}) T_{uv}^{(\text{out})}, \quad \mathbf{Y}^+ = \mathbf{F} + (\mathbf{Y}^+)^T \cdot \mathbf{T}^{(\text{out})}. \quad (3.1.45b)$$

figure

Figure 3.1: Illustration of average neighbour value node measures $Y_v^-(\mathcal{I})$ and $Y_v^+(\mathcal{F})$ for node v .

We can see these in terms of a diffusion process if we find the simple solution for these iterative definitions which we can use as an alternative definition

Definition 3.1.10— Average neighbour values Y_v^\pm

The centrality measure $Y_v^-(\mathbf{I})$ for node v is proportional to the sum of the probabilities that a random walker moving backwards leaving a node v will arrive (at any time) at each initial node i where $I_i = 1$. Similarly, the centrality measure $Y_v^+(\mathbf{F})$ for node v is proportional to the sum of probabilities that a random walker leaving a node v and moving forwards will arrive at each final node f where $F_f = 1$. That is, for general initial and final vectors, we have

$$Y_v^-(\mathbf{I}) = \sum_u \left[\sum_{n=0}^{\infty} (\mathbf{T}^{(\text{in})})^n \right]_{vu} I_u, \quad \mathbf{Y}^- = \left[\sum_{n=0}^{\infty} (\mathbf{T}^{(\text{in})})^n \right] \mathbf{I}, \quad (3.1.46a)$$

$$Y_v^+(\mathbf{F}) = \sum_u F_u \left[\sum_{n=0}^{\infty} (\mathbf{T}^{(\text{out})})^n \right]_{uv}, \quad \mathbf{Y}^+ = \mathbf{F}^T \left[\sum_{n=0}^{\infty} (\mathbf{T}^{(\text{out})})^n \right]. \quad (3.1.46b)$$

In a sense Y_v^- is measuring the influence that node v has on the initial nodes in \mathcal{I} while X_v^- measures the influence each initial node has on v . These are not the same values, as indicated by the fact that we use $\mathbf{T}^{(\text{in})}$ for Y_v^- but $\mathbf{T}^{(\text{out})}$ for X_v^- . Even if we were to define equivalent measures on a standard undirected graph (\mathbf{T}) ²² there would be no past/future versions but we would have a distinct X_v and Y_v values as we look two different directions for the influence, on v or from v .

²²(T)What is this equivalent measure?

SpringRank

We have seen how we can use broadcast processes through the adjacency matrix \mathbf{A} or node-driven diffusion processes defined by $\mathbf{T}^{(\text{out})}$ and $\mathbf{T}^{(\text{in})}$. In particular, one way to look some of these measures is that we take sums or averages of the values of our neighbours, either the child node or the parent nodes, but never both at the same time. That is we could consider a measure S_v for node v based on the average of the sum of all neighbours, both child and parent nodes. This averaging of both child and parent nodes to give a node value is what the SpringRank measure uses so let us look at this.

SpringRank aims to give put nodes in a directed network in some sort of order, so it is giving them a rating, a real valued score, such that a higher score is to be thought of as coming later than those nodes with lower scores, i.e. it defines a total order on the network. There is no reason why we cannot do this on a directed network with no cycles. Equally, there is no guarantee that the total order found from SpringRank will respect the partial order inherent in a DAG. This does not mean we should dismiss this measure out of hand for DAGs. All the measures considered so far, provide either a rank (an integer value) or a rating (a real value) to all nodes. In some cases, defining an order on the nodes may be intended (e.g. height and depth) but in others, such as those in ecological contexts, the measures were trying to identify important nodes in some appropriate sense for that context. So we will approach SpringRank as just another node measure. If it is useful for a given problem will have to come from the context.

If SpringRank were just an average of parent and child node SpringRank values, this measure would be uninteresting as one solution is obviously that $S_v = 0$ for all nodes v . In fact, as we will see below, by treating parent and child nodes equally, we lose all sense of order and direction in the DAG. So the definition of SpringRank also includes one other key element, namely that we add on a term equal to the difference between the in- and out-degree²³, normalised by the same factor as the average of child and parent nodes uses. What this extra term is doing is injecting some asymmetry at each node, using the local properties of the node to do this.

Definition 3.1.11— SpringRank I

The SPRINGRANK S_v of node v is the average of the SpringRank of the child nodes minus one plus the average of the parent nodes plus one.

$$S_v = \frac{1}{Z_v} \left(\left(\sum_u A_{vu} S_u \right) + \left(\sum_u S_u A_{uv} \right) - J_v \right), \quad (3.1.47)$$

$$Z_v = \left(\sum_u A_{vu} \right) + \left(\sum_u A_{uv} \right) = \sum_u (A_{vu} + A_{uv}) = k_v. \quad (3.1.48)$$

$$J_v = k_v^{(\text{in})} - k_v^{(\text{out})} \quad (3.1.49)$$

The normalisation is simply $Z_v = k_v = k_v^{(\text{in})} + k_v^{(\text{out})}$, the total degree in this unweighted case. This definition may seem a little contrived but the J_v term anticipates some of the mathematical properties we will see below. The reason we presented this form first is that it emphasises that at the heart of SpringRank is an average of the SpringRank values of all nearest neighbours in contrast to the averaging we saw in the previous examples.

(T)IS the simple low order solution **J?** NO **(end T)**

²³In this discussion we assume an unweighted DAG. Everything works in exactly the same way with weights except we use strength not degree, $s_v^{(\text{in})}$ for $k^{(\text{in})}$ and $s_v^{(\text{out})}$ for $k^{(\text{out})}$.

This first definition of SpringRank is not the form often seen in the literature. If we rearrange the equation further, we find we can express this in terms of the network Laplacian of the symmetrised network with symmetric adjacency matrix $\mathbf{S} = \mathbf{A} + \mathbf{A}^T$. That is if we define the diagonal matrix \mathbf{D} with total degree along the diagonal $D_{uv} = k_v \delta_{uv}$ then we have that

$$\sum_u \underbrace{(k_v \delta_{vu} - (A_{vu} + A_{uv}))}_{\text{network Laplacian of } \mathbf{A} + \mathbf{A}^T} S_u = k_v^{(\text{in})} - k_v^{(\text{out})}, \quad (3.1.50)$$

This then gives us the second way that this node measure is defined.

Definition 3.1.12—SpringRank II

The SpringRank S_v of a node v in a DAG is the solution to the equation

$$\mathbf{L} \cdot \mathbf{S} = \mathbf{J}, \quad \mathbf{L} = \mathbf{D} - \mathbf{A} + \mathbf{A}^T, \quad J_v = k_v^{(\text{in})} - k_v^{(\text{out})}, \quad (3.1.51)$$

where \mathbf{L} is the network Laplacian of the symmetric adjacency matrix $\mathbf{S} = \mathbf{A} + \mathbf{A}^T$.

It is intriguing that it is the symmetrised adjacency matrix that records the global structure of the network in this measure. The only place where the order and directionality survives is in the external source term \mathbf{J} but as this is based on the in- and out-degree, this is only sensitive to the directionality locally to each node. So a lot of the information on the order in a DAG has been lost.

The appearance of the Laplacian of the symmetrised adjacency matrix also reveals that this measure can be interpreted in terms of a stochastic process on the symmetrised network described by $\mathbf{S} = \mathbf{A} + \mathbf{A}^T$. That is we can look at values $s_v(t)$ defined at time t for node v which evolve according to

$$s_v(t+1) - s_v(t) = - \underbrace{\sum_u S_{uv} s_v(t)}_{\text{flow out of } v} + \underbrace{\sum_u S_{vu} s_u(t)}_{\text{flow into } v} + J_v \quad (3.1.52)$$

$$= - \left(\sum_u L_{vu} s_u(t) \right) + J_v. \quad (3.1.53)$$

This is an example of a broadcast process on the undirected network with adjacency matrix symmetrised $\mathbf{S} = \mathbf{A} + \mathbf{A}^T$ but one where conservation of the total value $S(t) = \sum_v s_v(t)$ is enforced, so not the broadcast process used in (1.4.1). That is we can show $S(t+1) - S(t) = \sum_v J_v$ where, from the definition of the external source/sink term \mathbf{J} , we can see that there is no net flow into or out of the network due to the \mathbf{J} term. However, here $= \sum_v J_v = 0$. The SpringRank measures are then the long-time limit of $s_v(t)$

$$s_v = \lim_{t \rightarrow \infty} s_v(t) \quad (3.1.54)$$

This gives us another way to interpret the SpringRank values. This is because the equation (3.1.53) represents properties of an electrical circuit (Doyle and Snell, 1984). The $s_v(t)$ is the voltage at each node v while the undirected edges (u, v) are resistors connecting u and v with conductance equal to the edge weight $S_{uv} = A_{uv} + A_{vu}$. Then J_v represents the current being added at each node, negative values represent current taken out of a node. The long-time limit represents the final equilibrium values and these are the SpringRank node values s_v .

There is then one last way to look at SpringRank. This is useful in terms of visualisation methods for DAGs or networks in general. So

Definition 3.1.13— SpringRank III

The SpringRank values S_v for nodes v in a DAG \mathcal{D} minimise the ‘energy’ $E(\mathcal{D})$ where

$$E(\mathcal{D}) = \frac{1}{2} \sum_{u,v} A_{vu}(s_v - s_u - 1)^2 \quad (3.1.55)$$

where \mathbf{L} is the network Laplacian based on the symmetric adjacency matrix $\mathbf{A} + \mathbf{A}^T$.

That is we can interpret s_v as the equilibrium position of node v on a line where for each directed edge (u, v) from node u to node v there is a perfect spring of spring constant A_{uv} but of natural length²⁴ 1 for all springs. The SpringRank is then the location of the nodes, the ends of the springs when there is no motion, the system is at a has minimum potential energy which is what $E(\mathcal{D})$ represents. For instance, if we had a line DAG, where every node had one incoming edge and one coming edge except for the ends, one sink and one source node, the solution is clearly where every node is one unit apart from its neighbours and the s_v increases with height, e.g. $s_v = h_v + c$ are the solutions for any real constant c . Note the directionality in (3.1.55) as $(s_v - s_u - 1)$ is not symmetric and neither is \mathbf{A} ; so if $\{s_v\}$ is one solution, the negative of this, $\tilde{s}_v = -s_v$, is not a solution..

To see that this third definition matches the previous one, we simply take derivatives with respect to each s_v and set them all to zero to find a minimum, $\partial E / \partial s_v = 0$.

$$\partial E / \partial s_w = \sum_u A_{vu}(s_v - s_u - 1)(\delta_{vw} - \delta_{wu}) \quad (3.1.56)$$

$$= \sum_u A_{wu}(s_w - s_u - 1) - \sum_v A_{vw}(s_v - s_w - 1) \quad (3.1.57)$$

$$= + \sum_u \delta_{uw}(k_w^{(in)} + k_w^{(out)}) - (A_{wu} + A_{uw})s_u - k_w^{(in)} + k_w^{(out)} \quad (3.1.58)$$

which is exactly (3.1.51) used in definition II.

The iterative form of node height and depth in (3.1.14) is suggests a generalisation

$$y^-(v, p) = 1 + \frac{1}{|[\mathbf{A}\mathbf{1}]_v|_p} \left(\sum_u A_{vu}(y^-(u, p))^p \right)^{1/p}, \quad |[\mathbf{A}\mathbf{1}]_v|_p = \left(\sum_u A_{vu} 1^p \right)^{1/p} = (k_v^{(out)})^{1/p} \quad (3.1.59)$$

$$y^+(v, p) = 1 + \frac{1}{|[\mathbf{1}^T \mathbf{A}]_v|_p} \left(\sum_u A_{uv}(y^+(u, p))^p \right)^{1/p}. \quad |[\mathbf{1}^T \mathbf{A}]_v|_p = \left(\sum_u 1^p A_{uv} \right)^{1/p} = (k_v^{(in)})^{1/p} \quad (3.1.60)$$

Here the $|[\mathbf{A}\mathbf{1}]_v|_p$ and $|[\mathbf{1}^T \mathbf{A}]_v|_p$ represent reasonable normalisations. The $p \rightarrow +\infty$ recovers the max formulation²⁵ used in height and depth definitions (3.1.14), so $h_v = y^-(v, p = \infty)$ and $d_v = y^+(v, p = \infty)$. The $p = 1$ recovers the probabilistic form used for trophic levels (T)²⁶.

²⁴We could replace this 1 in the expression by some constant a but all it does is set the length scale. We could remove a by working in terms of dimensionless variables s_v/a which is what we have chosen to do in the form (3.1.55).

²⁵This is provided we modify the normalisation. These should count the number of terms in the numerator which are equal to the maximum value, say $n^{(\max)}$, as then we need to normalise by $(n^{(\max)})^{1/p}$. Either that or we have to make sure we never get two or more terms in the numerator equal to the maximum value.

²⁶(T)We interpret $|[\mathbf{A}\mathbf{1}]_v|_p$ to be $\max_u(A_{vu})$ for $p = +\infty$. These normalisations are simply $(k^{(in)})^{1/p}$ and $(k^{(out)})^{1/p}$?

In particular for $p = 1$ we have that

$$y^-(v, 1) = 1_v + \left(\sum_u T_{vu}^{(\text{in})} y^-(u, p) \right), \quad \mathbf{y}^-(p) = \left(\sum_{n=0}^{\infty} (\mathbf{T}^{(\text{in})})^n \right) \mathbf{1}, \quad (3.1.61)$$

$$y^+(v, 1) = 1_v + \left(\sum_u y^+(u, p) T_{uv}^{(\text{out})} \right), \quad \mathbf{y}^+(p) = \mathbf{1}^T \left(\sum_{n=0}^{\infty} (\mathbf{T}^{(\text{out})})^n \right). \quad (3.1.62)$$

where

$$T_{vu}^{(\text{in})} = \frac{A_{vu}}{k_v^{(\text{in})}}, \quad T_{uv}^{(\text{out})} = \frac{A_{uv}}{k_v^{(\text{out})}}. \quad (3.1.63)$$

Here the value $y^-(v, 1)$ is the average of the parent node values. Equally y^- is following the paths of backwards/reverse random walkers so moving backwards into the past towards the sources. By way of comparison, the $x^-(v)$ measures also look to nodes in the past but they use random walkers moving forward from the past to the current node v .

(T) See p147 and table 1 p154 in Jordán and Scheuring (2004). We have also seen “modal path length” in Pimm and Lawton (1978), Pimm (1982), Jordán and Scheuring (2004) which is the mode (most common) length of all the paths from the nodes to the bottom (producers) in a food web. Unclear if shortest or longest path lengths used but only shortest appears defined in texts. “(2) the minimal path length or shortest chain ($SC = 1$ for species C in Fig. 1) equals the minimal number of links connecting the focal species to any of the sources; (3) the maximal path length or longest chain ($LC = 2$ for species C in Fig. 1) equals the longest directed path from sources to the focal species, i.e., the path containing the largest number of links; (4) the chain average ($CA = 1.66$ for species C in Fig. 1) is the mean of every possible different path lengths from the sources to the focal species.” **(end T)**

Mixed Trophic Impact DAG centrality

(T)*** THIS NEEDS TO BE ADAPTED *** **(end T)**

We have seen in section 1.3 that many standard network centrality measures look at the large time limit of a process. For instance eigenvalue centrality is based on the dominant values of \mathbf{A}^n as $n \rightarrow \infty$, the number of infinite paths passing through a node, but for a DAG this is always zero since there are no cycles, no walks of infinite length. That is $\mathbf{A}^n = 0$ for $n > H$ where H is the height of the DAG. Whichever node you start from in a DAG, if you follow the direction of the edges you will always reach the end of the DAG. One way round this problem is to look at measures that use contributions from walks of all lengths, such as the generalisation of eigenvalue centrality. (T)²⁷ However, another approach is to use a more complicated process where we mix both forwards and backwards steps within any one walk. So let us look at one well known example of this.

The MIXED TROPHIC IMPACT measure was defined by Ulanowicz and Puccia (1990) in the context of ecology and food webs (see section 5.3) though this was in turn inspired by the Leontief Input-Output models of macroeconomics. The value for each node again depends on the wider DAG but, unlike with the keystone measures K_v^\pm of ecology in section 3.1.3, mixed trophic impact values are sensitive to the structure of the whole DAG, not just to the past or future node sets. Like the keystone measures, we can motivate the expressions for mixed trophic impact in terms of a diffusion process, one that used the same matrices $\mathbf{T}^{(\text{out})}$ and $\mathbf{T}^{(\text{in})}$ used for the keystone measures.

The mixed trophic impact is often usually calculated in terms of a weighted DAG. That is we work with an adjacency matrix \mathbf{A} . When $A_{vu} > 0$ we say A_{uv} is the weight²⁸ of an edge

²⁷(T) WHERE DO WE DO THIS for DAGs?

²⁸We will assume zero values in the adjacency matrix correspond to no edge, and we will not consider signed networks where negative weights are allowed.

from node u to node v . We then define our diffusive matrices in terms of strength rather than degree so

$$T_{vu}^{(\text{out})} = \begin{cases} \frac{1}{s_u^{(\text{out})}} W_{vu} & \text{if } s_u^{(\text{out})} > 0 \\ 0 & \text{if } s_u^{(\text{out})} = 0 \end{cases} \quad \text{where } s_u^{(\text{out})} = \sum_v W_{vu}. \quad (3.1.64\text{a})$$

$$T_{vu}^{(\text{in})} = \begin{cases} \frac{1}{s_v^{(\text{in})}} W_{vu} & \text{if } s_v^{(\text{in})} > 0 \\ 0 & \text{if } s_v^{(\text{in})} = 0 \end{cases} \quad \text{where } s_v^{(\text{in})} = \sum_u W_{vu}. \quad (3.1.64\text{b})$$

Again, the entries of these matrices lie between 0 and 1, $0 \leq T_{vu}^{(\text{in})}, T_{vu}^{(\text{out})} \leq 1$. In particular, when node v is not a sink or source node, then the v -th row of $\mathbf{T}^{(\text{in})}$ and the v -th column of $\mathbf{T}^{(\text{out})}$ sum to one. Conversely, for columns or rows associated with either a source or sink matrices, these are zero²⁹.

The mixed trophic index can be motivated in the context of a food web as follows. Suppose we imagine that our nodes (species) are all interacting, exchanging value (information, energy or whatever) along the edges of the DAG. This is generally a highly non-linear process but if we look at small increase $r_v(0)$ in the value at node v we can imagine that we can look at such small increases an approximate the relationship in the values between nodes as a linear relationship. Thus we will imagine that the first-order impact on a child node v of a small zero-th order change $r_w(0)$ at node w we define to be a small positive amount $r_w^+(1) = T_{wv}^{(\text{out})} r_v(0)$. So the change $r_v(0)$ is spread between the children in proportion to the weights of the edges. **(T)**³⁰ This is similar to the process used to define past Keystone index $K_w^{(-)}$. If this was all we considered then the mixed trophic index of node v would only be linked to nodes in the future of v .

However, unlike keystone index, we will also consider another process happening at the same time. That is we will treat the same zero-th order perturbation $r_v(0)$ at node v as having a *negative* effect on those subordinates, (parent) nodes u , of our node v . That is we define another part of the first-order effect to be $r_u^-(1) = -r_v(0)T_{vu}^{(\text{in})}$. Note that this is like the backwards diffusion process we used for part of the keystone index $\mathbf{K}^{(+)}$ also used on food webs.

The difference from the Keystone index is that we will assume both this positive forwards diffusion and this negative backwards diffusion happen at the same time, completely unlike what we did with either broadcast or diffusion count measures. That is we define the total change in the value of node u at time $t = 1$ to be

$$r_u(1) = \sum_v T_{uv}^{(\text{out})} r_v(0) - r_v(0) T_{vu}^{(\text{in})} = \sum_v [\mathbf{T}^{(\text{out})} - (\mathbf{T}^{(\text{in})})^\text{T}]_{vu} r_v(0). \quad (3.1.65)$$

Note that for algebraic convenience, in the second form we have put the change at time $t = 0$ on the right for both terms but this means we need to put the *transpose* of $\mathbf{T}^{(\text{in})}$ as part of the matrix in the square brackets.

(T)³¹

²⁹So these are not true stochastic matrices where every row (or every column) must sum to one. **(T)**Have we included this concept somewhere, otherwise remove it. **(end T)**

³⁰**(T)**For instance suppose nodes are people in an organisation and the T_{vu} gives the fraction of hours node u works on a project run by a superior, node v . Then we can imagine that any improvement in the output from node u (say after some training giving u a new skill) will lead to improved credit given to the superiors of u . This benefit will be split between the superiors in proportion to the weight of the connection between node u and each superior.

³¹**(T)**Again we will imagine that W_{uw} represents the hours that subordinate w works on projects run by u . Now we might imagine that if a superior v improves their skills by $r_v(0)$ then node u has less need of their subordinates w so their values declines by $r_w^-(1)$.

The idea now is that first-order changes then trigger second-order changes as the system is intricately connected as represented by the DAG. However we motivate the form, the abstract equation used to define the t -th order changes is defined to be

$$r_u(t+1) = \sum_u [\mathbf{T}^{(\text{out})} - (\mathbf{T}^{(\text{in})})^T]_{vu} r_u(t) = \sum_u Q_{vu} r_u(t). \quad (3.1.66)$$

where we define a new matrix \mathbf{Q}

$$\mathbf{Q} = \mathbf{T}^{(\text{out})} - (\mathbf{T}^{(\text{in})})^T. \quad (3.1.67)$$

This matrix new matrix \mathbf{Q} represents a process which is a mixture of diffusion forwards and and backwards on the DAG as well as adding a negative value for flow backwards on the DAG. This is very different from the keystone index case where flows were only ever in one of the two directions when constructing one of the keystone indices.

Next we imagine that all nodes are equal and so we make a small but equal-sized change to all nodes³², $r_u(t=0) = 1$. Finally, just as we did with the broadcast and diffusion based counts, and with the keystone index, we then sum up all the subsequent changes the initial perturbation causes to get a total R_v at each node v and this \mathbf{S} is the MIXED TROPHIC INDEX. This That is

$$R_v = \sum_{t=1}^{\infty} r_v(t) = \sum_{t=1}^{\infty} \sum_u [\mathbf{Q}^t]_{vu} r_u(t=0) = \sum_{t=1}^{\infty} \sum_u [\mathbf{Q}^t]_{vu} \quad (3.1.68)$$

We can write out \mathbf{Q} as

$$Q_{vu} = \begin{cases} \frac{1}{s_u^{(\text{out})}} A_{vu} - \frac{1}{s_v^{(\text{in})}} A_{vu} & \text{if } s_u^{(\text{out})} > 0 \text{ and } s_v^{(\text{in})} > 0 \\ \frac{1}{s_u^{(\text{out})}} A_{vu} & \text{if } s_u^{(\text{out})} > 0 \text{ and } s_v^{(\text{in})} = 0 \\ -\frac{1}{s_v^{(\text{in})}} A_{vu} & \text{if } s_u^{(\text{out})} = 0 \text{ and } s_v^{(\text{in})} > 0 \\ 0 & \text{if } s_u^{(\text{out})} = 0 \text{ and } s_v^{(\text{in})} > 0 \end{cases} \quad (3.1.69)$$

where $s_v^{(\text{out})}$ and $s_v^{(\text{in})}$ are the in- and out-strength of node v respectively. We can now give the definition of the mixed tropic index.

Definition 3.1.14— Mixed Trophic Index

The MIXED TROPIC INDEX for node v is simply the sum of the entries of the v -th column of the matrix $\mathbf{M} = \sum_{t=1}^{\infty} \mathbf{Q}^t$.

$$R_v = \sum_{t=1}^{\infty} \sum_u [\mathbf{Q}^t]_{vu} \quad (3.1.70)$$

In the end, we are just assigning a score to each node. Though we motivated it in terms of a small perturbation in the initial values at nodes, these are node made in proportion to, say, the biomass of each species. We are not looking at the final equilibrium state which might have some ecological meaning. Rather we are scoring each node based on how the perturbations

³²Again, if you have more information on nodes, this is something you could change to give more weight to initial changes in some nodes than others through $r_u(t=0)$ and that would lead to a useful variation in the index. This would be the equivalent of the \mathbf{I} and \mathbf{F} vectors used in the broadcast and diffusion path count measures.

cause changes at each node at *all* times. If all the changes were positive, this would simply diverge. However, the negative sign used for the backwards flows mean that mixed trophic index can converge to a answer which is found to be useful, in ecology at least.

To see that we can get a convergent results, let us assume that our DAG is a single weakly connected component with at least one edge. Then the last case in (3.1.69) does not occur as it represents an isolated node with no edges. Now the \mathbf{Q} matrix represents a strongly connected weighted if *signed* directed network. The key point is because of the reverse DAG terms coming from $\mathbf{T}^{(\text{in})}$ we can go forwards and backwards in the original DAG so there are paths in both directions between all nodes. In terms of mathematics, this means we can (almost) always write $\sum_{t=0}^{\infty} \mathbf{Q}^t = [\mathbf{1} - \mathbf{Q}]^{-1}$, i.e. our answer can be written in terms of the inverse of a matrix $[\mathbf{1} - \mathbf{Q}]$.
(T) One way to see this is **(end T)**

Thus we have that

$$R_v = \sum_u M_{vu} \quad \text{where} \quad \mathbf{M} = \sum_{t=1}^{\infty} \mathbf{Q}^t = (\mathbf{1} - \mathbf{Q})^{-1} - \mathbf{1}. \quad (3.1.71)$$

We have already highlighted a similarity between the keystone measure and the mixed trophic index. Both are written in terms of two transfer matrices, $\mathbf{T}^{(\text{in})}$ and $\mathbf{T}^{(\text{out})}$ with the weighted network versions used for mixed trophic index in (5.3.6) and (5.3.7) reducing to the unweighted versions in (5.3.4) and (5.3.5). Then we note that the \mathbf{M} matrix used to define mixed trophic index is also

$$R_u = \left[\sum_v \sum_n \mathbf{Q}^n \right]_{vu} = \left[\sum_v \sum_n [\mathbf{T}^{(\text{out})} - (\mathbf{T}^{(\text{in})})^T]^n \right]_{vu} \quad (3.1.72)$$

$$= \left[\sum_v \sum_n [\mathbf{T}^{(\text{out})}]^n \right]_{vu} + \dots + \left[\sum_u \sum_n (-1)^n [(\mathbf{T}^{(\text{in})})^T]^n \right]_{vu} \quad (3.1.73)$$

$$= K_v^{(-)} + \dots + \left[\sum_u \sum_n (-1)^n [(\mathbf{T}^{(\text{in})})^T]^n \right]_{vu} \quad (3.1.74)$$

So the first term in the \mathbf{M} used for mixed tropic index is just the $K_v^{(-)}$ keystone measure. The last term is almost $K_v^{(+)}$ except that there is a sign difference between odd and even powers of $\mathbf{T}^{(\text{in})}$. So in the end there is a significant difference in the algebraic formulae for these two ecological measures despite the fact they are both based on the same diffusion processes.

To round off this discussion of mixed trophic index, we have noted that in the context of ecology, certain choices have been made. These could be changed. For instance we could start the process from different perpurbations $\mathbf{r}(0)$. The actual process could be changes too. We could modify the weights given to the forwards and backwards processes, using $\alpha\mathbf{T}^{(\text{out})} + \beta\mathbf{T}^{(\text{in})}$ though convergence will put limits on the choices for the parameters α and β . We could even further and return to a broadcast process, replacing $\mathbf{T}^{(\text{out})}$ by \mathbf{A} and $\mathbf{T}^{(\text{in})}$ by \mathbf{A}^T to get an even large family of generalisations of broadcast path count measures. We will leave all these generalisations at this point and instead turn to a different way to combine forwards and backwards processes on a DAG in order to analyse the DAG.

Forwards-backwards centrality

(T)*** Should we swap time t for τ as we use t for target or sink nodes? **(end T)**

We commented when introducing the mixed trophic index that using a process with both forwards and backwards steps on a DAG gets round the problem that walks in one direction of a DAG are finite as they cannot have more steps than the height of the DAG. The Mixed Trophic Impact centrality measure does exactly this, mixing forwards and backwards steps in a diffusion process encoded in $\mathbf{Q} = \mathbf{T}^{(\text{out})} - \mathbf{T}^{(\text{in})}$ of (3.1.67). So at each node in a walk we extend

the walk by looking at walks going forwards half the time and walks going backwards half the time.

We could use this idea of using both forwards and backwards processes in another way, by moving forwards on one step and then backwards on the next, repeating as needed. That is we could work with a process based around diffusion but where we first go forwards and then the next step is backwards, so based around $\mathbf{K} = \mathbf{T}^{(\text{in})} \cdot \mathbf{T}^{(\text{out})}$. Equally we could look at a forwards-backwards broadcast process using

$$\mathbf{H} = \mathbf{A}^T \cdot \mathbf{A}. \quad (3.1.75)$$

This second case is the basis of a well known centrality method used on directed networks, HYPERLINK-INDUCED TOPIC SEARCH (Kleinberg, 1999) usually known by its abbreviation **HITS**. We will focus on the broadcast-based HITS but the concepts are the same for diffusion forwards-backwards process \mathbf{R} .

For HITS we can define two values, the AUTHORITY VALUE X_v and the HUBS VALUE Y_v , for each node v . These are defined through the following processes

$$x_v(t+1) = \sum_u A_{vu} y_u(t), \quad (3.1.76a)$$

$$y_v(t+1) = \sum_u A_{uv} x_u(t). \quad (3.1.76b)$$

It is crucial to note that in the definition of the hub values $\mathbf{y}(t+1)$ in (3.1.76b), the transpose of the adjacency matrix is used so the value $x_u(t)$ is copied to the predecessors/parents of u . Put another way the authority values $\mathbf{x}(t)$ flow backwards along the edges to create the next hub values $\mathbf{y}(t+1)$, while the next authority values $\mathbf{x}(t+1)$ come from the previous round of hub values $\mathbf{y}(t)$ flowing forward along the direction of the edges. We can show this by combining (3.1.76a) and (3.1.76b) as

$$x_v(t+2) = \sum_{u,w} A_{wv} A_{wu} y_u(t) = \sum_u H_{wu} y_u(t), \quad \mathbf{x}(t+2) = \mathbf{H} \mathbf{x}(t), \quad (3.1.77a)$$

$$y_v(t+2) = \sum_{u,w} A_{vw} A_{uw} y_u(t) = \sum_u H_{uw} y_u(t), \quad \mathbf{y}(t+2) = \mathbf{H}^T \mathbf{y}(t). \quad (3.1.77b)$$

Since we are going forwards then backwards, the effective network with adjacent matrix $\mathbf{H} = \mathbf{A}^T \cdot \mathbf{A}$ is weighted, directed, has cycles even if \mathbf{A} describes a DAG as we assume here. If we have a sink node s in the DAG, then $A_{vu} x_u = 0$ for any v as there are no outgoing edges. However, all other types of nodes, including source nodes, always have at least one outgoing edge described in \mathbf{A} . So when we take the second reverse step described by \mathbf{A}^T we can always return to the origin node. That is, apart from sink nodes, all node in the network described by \mathbf{H} have self-loops. Thus \mathbf{H} projects down on to a space containing all nodes except the sink node. In the same way the matrix needed for the hubs values, $\mathbf{A} \mathbf{A}^T$ projects down on the set of nodes excluding the source nodes.

Since there is no sense of conservation associated with the broadcast process, these node values tend to grow, controlled by the largest eigenvalue of the matrix $\mathbf{H} = \mathbf{A}^T \cdot \mathbf{A}$. For that reason, the centrality values are defined to be normalised versions of these long-time limits, authority values X_v from $\mathbf{x}(t \rightarrow \infty)$ and hub values Y_v from $\mathbf{y}(t \rightarrow \infty)$. Any normalisation will do but for definiteness we can define

$$X_v = \lim_{t \rightarrow \infty} \frac{\mathbf{x}(t)}{|\mathbf{x}|_1(t)}, \quad |\mathbf{x}|_1 = \sum_v |x_v|, \quad (3.1.78a)$$

$$Y_v = \lim_{t \rightarrow \infty} \frac{\mathbf{y}(t)}{|\mathbf{y}|_1(t)}, \quad |\mathbf{y}|_1 = \sum_v |y_v|. \quad (3.1.78b)$$

The discussion about sources and sinks show that, assuming we start at $t = 0$, then $x_v(t > 0) = 0$ for any sink node v and $y_v(t > 0) = 0$ for any source node v . So we know that sink nodes $t \in \mathcal{T}$ have zero authority score $X_t = 0$ and source nodes $s \in \mathcal{S}$ have zero hubs score $Y_s = 0$.

If we want to see how this works in more detail, we can write the solution for $\mathbf{x}(t)$ in terms of the eigenvectors of \mathbf{H} since the solution is of the form

$$\mathbf{x}(t) = \sum_n (\lambda_n)^t c_n \mathbf{v}^{(n)} \quad \text{where} \quad \mathbf{H}\mathbf{v}^{(n)} = \lambda_n \mathbf{v}^{(n)}, \quad (3.1.79a)$$

$$c_n = \sum_i x_i(t=0) v_i^{(n)} / |\mathbf{v}^{(n)}|. \quad (3.1.79b)$$

The c_n are set from the initial conditions assumed to be at $t = 0$. For most DAGs, the adjacency matrix \mathbf{A} represents a DAG where either or both in- and out-strengths of nodes is one or more. In these cases the eigenvalue of \mathbf{H} with largest absolute value, say λ_1 , is both real and is greater than one³³ It then follows that the values for $\mathbf{x}(t)$ come to be dominated by the leading eigenvector associated with the largest eigenvector, but scaling by $(\lambda_n)^t$. Similar arguments apply to the $\mathbf{y}(t)$ values.

Example 3.1.1 Hub and Authority values for the line DAG

Consider the line DAG with three nodes with edges from node 1 to node 2 and from node 2 to node 3. Then we have that

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \Rightarrow \quad \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{A}\mathbf{A}^T = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (3.1.80)$$

The matrix \mathbf{H} for the authority scores has a zero eigenvalue corresponding to the single sink node 3, so eigenvector $(0, 0, 1)^T$ has eigenvalue 0. The other two eigenvectors can be chosen to be $(1, 0, 0)^T$ and $(0, 1, 0)^T$ both have eigenvalue 1. This is not exactly what we stated in the main text as \mathbf{H} represents a network or more than one component even after the sink nodes are ignored. This pushes us outside of the strict conditions of the Perron-Frobenius theorem. However, by working on each strongly connected component separately, we arrive at the same conclusions as the main text. Namely that taking the normalised long-time limit of this process delivers a well defined centrality measure. So the authority values are 1/2 for nodes 1 and 2, and zero for node 3. A similar analysis shows the hubs values are 0 for node 1 and 1/2 for nodes 2 and 3.

3.1.4 Coda: Comments on node centrality in DAGs

Many variations but key aspects are

- Local vs global.
- single vs multiple path

³³This follows if we focus on the nodes excluding the sink nodes. If we look at the part of the \mathbf{H} matrix defined on those nodes, say $\tilde{\mathbf{H}}$, this a non-negative matrix where the rows and columns always add to at least one. Using the Perron-Frobenius theorem on this we arrive at our conclusion about most DAGs. It is possible to break this. The simple example of the line DAG has $\lambda_1 = 1$. If the original DAG has edge weights less than 1.0, it is easy to ensure $0 < \lambda_1 < 1$. However, for most DAGs encountered, $\lambda_1 > 1$. For these other cases, we can adapt the analysis in the main text without too much trouble. Note this same reasoning applies to eigenvalue centrality for standard networks which uses $\lim_{n \rightarrow \infty} \mathbf{A}^n$.

- single path distance choice
- multiple path, spreading/process matrix
- respecting direction, going backwards, or mixing direction
- Linear algebra, measures based on processes which are linear and Markovian (but not always conserved).
- Long time limit vs sum of effects over all times.
- Processes where values are non-negative, or not.
- normal/TR/TC DAG.
- Invariance under TR/TC
- dependence on size of past/future connection kernels of their product, use maximum bounds to normalise e.g. (Gualdi et al., 2011).

3.2 Edge and Path analysis for DAGs

(T)TSE criticality analysis here? (**end T**)

3.2.1 Longest path analysis

We established that we could assign a height and depth value to every node in section 2.5. The height $h(v)$ is the largest network length of any path from any source to a node v while the depth $d(v)$ is the largest network length of any path from that node to any sink. Here we will think about these measures in the same terms as we are doing for other node centrality measures. That is while they are designed

First, we can see these are similar to (T)?????? (**end T**)

there is a neat property that the sum of the height and depth for a node equals the height of the DAG $H = h(\mathcal{D})$ if and only if the node lies on at least one of the paths that define the height of the DAG. We can see this as follows. The height $h(v)$ for some node v is defined by using a path, say $P_h(v)$ from a source node to that node v . In the same way the depth $d(v)$ for the same node v is defined using a path $P_d(v)$ from that node v to a sink node. If we combine these two paths then we have a path from a sink to a source via v that of length equal to $h(v) + d(v)$. The values for $h(v)$ and $d(v)$ are as large as possible by the definition of height and depth so $h(v) + d(v)$ cannot be any larger. If this value is not equal to the height then this path cannot be one of the set of longest paths in the DAG, those of length L used to define the height of the DAG H . Equally, this value cannot be bigger than H as this would contradict the definition of the height of the DAG. So by definition of the height of the DAG, if this combined path $P_h(u) \cdot P_d(u)$ is of length $h(v) + d(v) = H$ it is an example of one of the longest paths in the DAG which are of length H . Sometimes this is recast as the CRITICALITY $c(v)$ of a node v (Ho et al., 2024a) where

$$c(v) = H - h(v) - d(v). \quad (3.2.1)$$

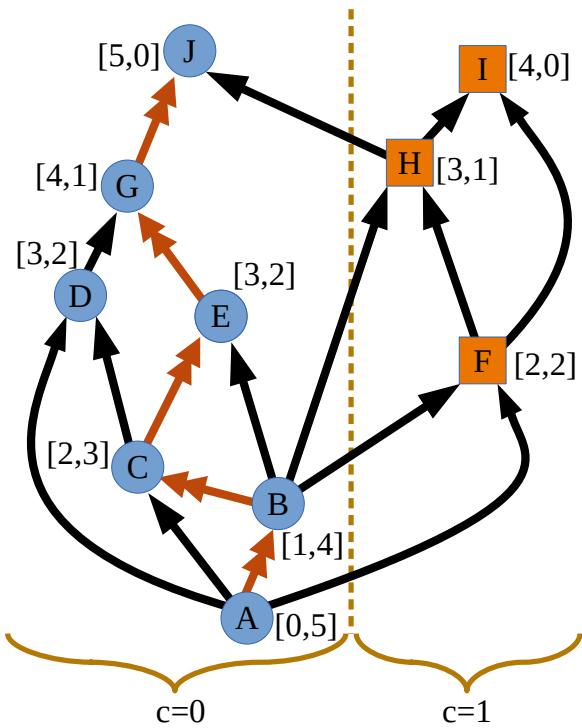


Figure 3.2: An illustration of height, depth and criticality of nodes in the same DAG as used in Fig. 2.7. Beside each node v the height and depth of that node are given as $[h(v), d(v)]$.

The double headed red arrows show an example of one of the longest paths in this DAG, namely (A, B, C, E, G, J) , of length 5. Thus the height this DAG, is $H = 5$.

The criticality $c(v) = H - h(v) - d(v)$ (3.2.1) of the nodes in blue circles on the left is zero so these all lie on at least one of the longest paths that define the height H of the DAG. The nodes in orange squares on the right have criticality 1.

Note that the way we have laid out the nodes also illustrates how we can use criticality and height to set the rough coordinates of nodes to arrive at a reasonable visualisation of a DAG.

3.2.2 Main path analysis

(T) Develop first as node centrality. Show on DNA as node pair values. Then show as edge values and MPA in later section. (end T)

(T) Critique of MPA: lack of theoretical basis, few algorithms. (end T) A fast algorithm was laid out in Batagelj (2003). Presumably, the implementation in Pajek uses this approach.

(T) There are publicly available algorithms but unclear if any of these are well maintained and well documented. A search for “main path analysis” on GitHub threw up several including:-

- <https://github.com/jfb-h/MainPaths.jl>, julia, May 2023.
- https://github.com/felipefrr/mpa_splc, python, December 2022.
- <https://github.com/bencabrera/wikiMainPath>, c++ web app, November 2018.

However, of these public ones with some code present (others had no code), none of the above looked to be active. (end T)

One of the main tools used when exploring the flow of information and knowledge in a citation network is known as MAIN PATH ANALYSIS (MPA). This was invented by Hummon and Doreian (1989) in the context of a dataset looking at the history of the discovery and modelling of DNA based on forty events (often dated by a publication) (Garfield et al., 1964) as shown in Fig. 3.3. (T)³⁴ Main path analysis has proved to be very popular (see Liu et al. (2019) for an overview). (T)³⁵ (T) Find papers and show visually? (end T) (T) Fill in more details if needed. (end T)

³⁴(T)(references here, TSE has a copy of this data)

³⁵(T)For example see (Hummon and Doreian, 1989, Carley et al., 1993, Batagelj, 2003, Verspagen, 2007, Lucio-Arias and Leydesdorff, 2008, Fontana et al., 2009, Liu and Lu, 2012, Yeo et al., 2013, Cerinsek and Batagelj, 2014, Xiao et al., 2014, Batagelj et al., 2014, Liu and Kuan, 2015, Barbieri et al., 2017, Filippin, 2021, Kim et al., 2022) . Maybe look at the following web sites: pajek, garfield, sussex, da Vincier Lab Main Path web page.

Main path analysis is usually used in the context of citation networks where the DAGs is unweighted. So we will consider this case where the adjacency matrix of the DAG contains only ones or zeros³⁶.

Main path analysis on any DAG is defined in two steps (for example see Liu et al. (2019), Kuan (2022)). First, each edge is given a value, the edge weight G_{vu} , equal to the number of paths passing through the edge (u, v) . These paths are all the paths that run between a set of starting nodes \mathcal{I} and a set of finishing nodes \mathcal{F} . Different variants of main path analysis correspond to different choices for these initial and final nodes and the paths used. In the second step, we look for the path with the highest weight, where the weight of a path $(\mathbf{T})^{37}$ is the sum of the edge weights of the edges in the path. The path with the largest total weight is the MAIN PATH. So this is a type of optimal path analysis but on a weighted version of the original DAG. $(\mathbf{T})^{38}$

Given that the main path analysis is based on the number of paths passing through an edge it should be no surprise we can rewrite this in terms of the node broadcast path count centrality measures \mathbf{W}^- and \mathbf{W}^+ of (??) that count the number of paths arriving at or leaving from a node. So we can use $W_v^- \equiv W^-(\mathcal{I})$ that counts all paths to v from any node i in the initial node set \mathcal{I} while $W_v^+ \equiv W^+(\mathcal{F})$ is the number of paths from v to any node f in the final node set \mathcal{F} . This is because the number of paths G_{vu} passing along any edge (u, v) is equal to the number of paths W_u^- arriving at the source node u of the edge multiplied by the number of paths W_v^+ leaving the target node v of the edge. Every path arriving at u can be extended by following any one edge from u to v and then extended further by adding all the nodes of any path leaving v and this gives a well defined path. So we can write the edge weights G_{vu} needed for the main path analysis as $(\mathbf{T})^{39}$

$$G_{vu}(\mathcal{F}, \mathcal{I}) = W_v^+(\mathcal{F}) A_{vu} W_u^-(\mathcal{I}). \quad (3.2.2)$$

$$G_{vu}(\mathbf{F}, \mathbf{I}) = [\mathbf{F} \cdot \sum_{n=0}^{\infty} (\mathbf{A})^n]_v A_{vu} [\sum_{m=0}^{\infty} (\mathbf{A})^m \cdot \mathbf{I}]_u. \quad (3.2.3)$$

We put a factor of A_{vu} here so this is zero if there is no edge from u to v and this ensures we have a weighted DAG described by the adjacency matrix \mathbf{G} . This weighted DAG has the same edges as the original DAG described by the adjacency matrix \mathbf{A} . Also note that this tells us that numerically, we can find the weights in this new weighted DAG \mathbf{G} very quickly once we have the broadcast centrality measures and they, in turn, can be found very quickly.

The final step is to find the path with the highest weight in the network described by \mathbf{G} . Again, standard numerical algorithms to find the longest network path work just as quickly when edges have weights so this is as straightforward as finding the paths with greatest network length (i.e. when edges with weight 1).

The one problem with this algorithm numerically is that the number of paths can grow extremely fast (often faster than exponential) as we move away from an initial node. $(\mathbf{T})^{40}$ This is made worse when we take a product of these values to find the edge weights G_{uv} . This usually means that we need to look outside standard built in methods used in most programming languages for storing integers.

One possible solution to the large values of G_{vu} is to use logarithms. That is we can define the edge weights using a PATH ENTROPY measure, simply the logarithm of the number of

³⁶In fact, the algebra also works for weighted DAGs without change but the language of the main text will reflect our unweighted DAG assumption.

³⁷ (\mathbf{T}) (the “traversal count”)

³⁸ (\mathbf{T}) Push later or to appendix? Approximations are sometimes used leading to further variants (Hummon and Doreian, 1989, Liu and Lu, 2012, Yeo et al., 2013, Liu and Kuan, 2015).

³⁹ (\mathbf{T}) Surely some functional form here?

⁴⁰ (\mathbf{T}) Need to note this for the broadcast count discussion

paths. So, in this form we work with edge weights $G_{vu}^{(H)}$ (now real values) for edges (u, v) where $G_{vu}^{(H)} = \ln(G_{vu})$ so

$$G_{vu}^{(H)} = \ln(G_{vu}) = \ln(W_v^+) + \ln(W_u^-). \quad (3.2.4)$$

This path entropy approach has the advantage that the weight $G^{(H)}(\mathcal{P})$ of a path \mathcal{P} is simply (assuming $A_{vu} = 1$)

$$G^{(H)}(\mathcal{P}) = \sum_{v \in \mathcal{P}} \ln(W_v^-) + \sum_{v \in \mathcal{P}} \ln(W_v^+) = \sum_{v \in \mathcal{P}} \ln(W_v^- \cdot W_v^+) \quad (3.2.5)$$

That is the edge weights $G_{vu}^{(H)}$ are simply the sum of the logarithms of the total number of paths passing through each node on a path, which in turn is the entropy $H_v = \ln(W_v^- \cdot W_v^+)$ associated with the paths passing through a node v . So nodes with large values of the product $W_v^- \cdot W_v^+$ are likely to be important in main path analysis. So this entropy path measure is finding the path with maximum entropy. That is if we were to pick a path uniformly at random from the set of all possible paths between nodes in the initial node set \mathcal{I} to node in the final node set \mathcal{F} , you are most likely to pass through the nodes on the main entropy path. Put another way, from an path entropy and path information point of view, the main path is most likely to represent a path chosen at random. If we go back to thinking about the broadcast path count centrality measures, this main path discussion gives us motivation to look at the product rather than the sum of the broadcast path count node centrality measures.

(T)???? REACHED HERE *** !!!! (end T)**

We have emphasised in several places that the broadcast process is not the only type of process common to see in networks. So we can easily imagine moving to to a diffusion process, replacing \mathbf{A} by $\mathbf{T}^{(in)}$ and $\mathbf{T}^{(out)}$ as appropriate, say

$$H_{vu}(\mathcal{F}, \mathcal{I}) = X_v^+(\mathcal{F}) T_{vu}^{(out)} X_u^-(\mathcal{I}). \quad (3.2.6)$$

$$H_{vu}(\mathbf{F}, \mathbf{I}) = [\mathbf{F} \cdot \sum_{n=0}^{\infty} (\mathbf{T}^{(out)})^n]_v T_{vu}^{(out)} [\sum_{m=0}^{\infty} (\mathbf{T}^{(out)})^m \cdot \mathbf{I}]_u. \quad (3.2.7)$$

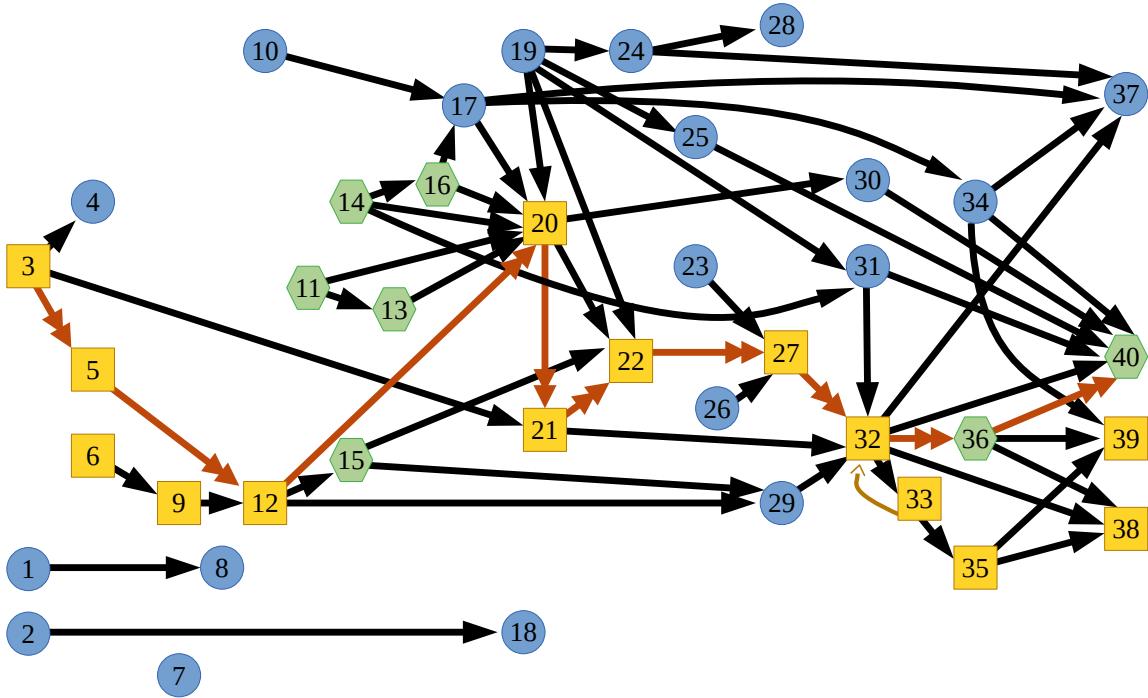


Figure 3.3: The historiogram (mostly a citation network) used in Hummon and Doreian (1989) to illustrate Main Path Analysis.main path analysis The data is drawn from the discussion by Asimov (1962) of the history of DNA. The nodes are mostly single documents and the edge indicated where Asimov suggests there is a direct connection in the earlier work leading to the later event. This is based on Fig.1 of Hummon and Doreian (1989) but to create a DAG we ignore the edge from node 33 (dated 1956) to node 32 (dated 1955), here indicated by a thin yellow line. The main path identified in Hummon and Doreian (1989) is indicated by red double headed arrows. However, note that we could replace node 40 with either node 38 or node 39 and still find a main path with the same score. The remaining edges in the DAG are black single headed arrows. The nodes in yellow squares have zero criticality, that is their height and depth add to be 10, the height of this DAG. criticality The nodes indicated by green hexagons have criticality 1, i.e. the longest path they lie on has length 9, one less than the height of the DAG. The blue circles are the remaining nodes with criticality more than one. The each node is labelled with the index used in Hummon and Doreian (1989). One topological order is given by the numerical order of the node index. The node index almost always puts the nodes in chronological order and this is used to order the nodes from left to right in the visualisation

3.3 The distance between nodes

- Theoretical framework: Distance, metric, pseudo/Lorentzian distances, reverse metric
- Network length - shortest and longest path
- Height/depth.
- Weighted path lengths
- The LOWEST COMMON ANCESTOR distance.

Terminology and notation

- “Network length”, “network distance” — length of a path in terms of the number of edges in that path.
- “Path weight” “weighted length” — length of a path as the sum of the weights in the path
- $d_{uv}^{(sp)} \equiv d^{(sp)}(u, v)$ — “shortest path distance” — network length of shortest path from/to ??? u to/from v.
- $d_{uv}^{(lp)} \equiv d^{(lp)}(u, v)$ — “longest path distance” — network length of shortest path from/to ??? u to/from v.
- $d(u, v)$ — “distance” — from/to ??? u to/from v???
- “distance”, “Lorentzian distance” (???) etc

3.3.1 The formal definition of distance

There is a formal definition of distance used for analytical work in mathematics. It is also useful for a wider audience in that the formal definition highlights fundamental properties that we expect to see in any definition of distance we use. For instance, we will see that the network length of the shortest path $d_{vu}^{(sp)}$ on undirected networks satisfies these criteria and the criteria help to reassure that our intuitive use of this measure is on firm ground. Equally, we will also see that the network length of the longest path $d_{vu}^{(lp)}$ on a DAG does *not* satisfy these criteria and that prompts us to think more carefully about the concept of the distance between nodes in a DAG.

In mathematics DISTANCE is simply a function d which assigns a real value to any pair of objects, say x and y , drawn from a given set \mathcal{X} , so $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. To be a distance this function must also satisfy three conditions (Deza and Deza, 2009): distances are never negative, the distance from x to y is the same as the distance from y to x , and finally the distance from a point to itself must always be zero. Formally these conditions are written as

$$d(x, y) \geq 0, \quad \text{non-negativity}, \quad (3.3.1a)$$

$$d(x, y) = d(y, x), \quad \text{symmetry}, \quad (3.3.1b)$$

$$d(x, x) = 0, \quad \text{reflexivity}. \quad (3.3.1c)$$

In our context, we are interested in the distances between nodes so that here $\mathcal{X} = \mathcal{V}$. Consider then a distance function equal to the length of the shortest path $d_{vu}^{(sp)}$ from any node u to any node v , that is $d_{u,v} = d_{vu}^{(sp)}$, then we see this does satisfy the properties (3.3.1) of a

formal ‘distance’ function. We do need to include trivial paths of just a single node and so of length zero for the reflexivity criteria to hold.

(T)We may not need this paragraph. We can take this formality a step further and see that our shortest path distance $d_{vu}^{(sp)}$ is also what is called a METRIC in mathematics. A SEMI-METRIC in mathematics is a distance function that also satisfies the TRIANGLE IDENTITY, that is

$$d(z, x) \leq d(z, y) + d(y, x) \quad \forall x, y, z, \in \mathcal{X}. \quad (\text{triangle inequality}). \quad (3.3.2)$$

If we want to finish this job, then the only difference between a metric and a semi-metric is that for a metric we tighten the reflexivity condition (3.3.1c) to become the identity of indiscernibles where

$$d(x, y) = 0 \text{ iff } x = y, \quad \text{identity of indiscernibles.} \quad (3.3.3)$$

Our network distance $d_{vu}^{(sp)}$ satisfies this criteria for a metric in a trivial manner as by definition we use the shortest path to define the distance from node u to v . If there was a path via some node k , the length of that path would be $d_{jk}^{(sp)} + d_{ki}^{(sp)}$. This path via k cannot be shorter than the path used to define the network distance $d_{vu}^{(sp)}$ as we define this to be the length of the shortest path. It may well be that this path via k is distinct and yet have the same length $d_{vu}^{(sp)}$ (there are often more than one shortest path of the same length between nodes) but this does not break the triangle identity. **(end T)**

Geodesic

(T)Do we need this? Perhaps make optional? (end T)

A final mathematical concept that is useful here is that of a GEODESIC which is a path whose length is a local minimum⁴¹. That is if we perturb our path by a small amount, we can only ever get a longer curve.

This is normally done in a space with continuous set of points \mathcal{X} where we can make infinitesimal changes to a path. For a network, it is much harder to define small changes. We can though imagine changing our path by a node or two. Suppose we have a path $\mathcal{P} = (v_0, v_1, \dots, v_L)$ and for some pair of consecutive nodes on this path, say v_n and v_{n+1} , they form a triangle with a distinct node u , that is there are also edges (v_n, u) and (u, v_{n+1}) . We can consider a new path $\mathcal{P}' = (v_0, v_1, \dots, v_n, u, v_{n+1}, \dots, v_L)$ but clearly this is longer than the original path. Another situation we could consider is where we find a path that routes via one different node, say a new path where v_n is replaced by u , so now $\mathcal{P}' = (v_0, v_1, \dots, v_{n-1}, u, v_{n+1}, \dots, v_L)$. If this exists it is a small change from the original path. However it is still the same length so provided we interpret “minimum” to allow for this case, alternative similar paths of the same length, we are still alright. Finally we could consider cases where we delete one node, say v_n from the shortest path, to find another path, so now $\mathcal{P}' = (v_0, v_1, \dots, v_{n-1}, v_{n+1}, \dots, v_L)$. This cannot happen as this path \mathcal{P}' would be shorter yet we started by assuming \mathcal{P} had the length of the shortest path and so a contradiction.

Usually our concepts of distance come from our natural interpretation of distance the real world. So the quickest route between two points is the shortest route where, if there are no

⁴¹Some discussions allow this to be simply stationary, i.e. a local minimum or a maximum. Some care is needed as there can be multiple geodesics between two points and sometimes they can be of different lengths. Consider a sphere where any two points always lie on a great circle and the points split that great circle into two arcs, one smaller than the other (ignoring the special case where they lie on opposite sides of the great circle). By this definition of a geodesic, both arcs are geodesics as any small local deviation of either arc leads to longer curves. Of course the global minimum, the one we would normally think of as the geodesic in most applications, is the short arc along the great circle.

obstacles, no hills, we measure distance using Euclidean distance⁴². The shortest path in these simple cases is always a straight line and that is the geodesic between those two points in this case. We can make a formal link between our use of these concepts in networks and our everyday interpretation of these concepts using a simple model. Consider a model where the nodes are thrown down uniformly at random in a Euclidean space, a process known as a POISSON POINT PROCESS — PPP. Suppose we then add edges between nodes only if that pair of points is within Euclidean distance R . Such a network is known as a RANDOM GEOMETRIC NETWORK, an RGG (Penrose, 2003). Then one may show rigorously that the shortest path in terms of the network path length has the shortest length if we add the Euclidean distances separations between neighbouring points in the path. That is the shortest path in the network is the best approximation to the straight line between the start and end point, the GEODESIC in this Euclidean space, for instance see (Díaz et al., 2016) and references therein. We will return later to the Poisson point process when looking at models for DAGs in section 4.

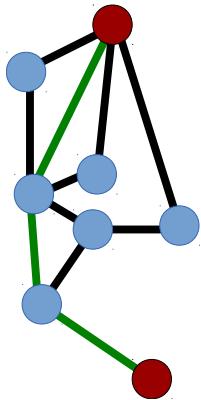


Figure 3.4: The shortest path between the highest and the lowest nodes, the two red nodes, is shown using the green edges. In this case there is a single path which has the shortest length between these two nodes, something which is not always true. If we look at all possible pairs of distinct nodes in this network, we will also see that 3 is the largest value of the shortest path length between any node pairs though there are several pairs which are three steps apart. This length, the largest of all the shortest path lengths, is the definition of the DIAMETER of a network.

3.3.2 Distances in DAGs

We will now look through some of the options we have in a DAG to measure the distance between nodes.

- Network length of shortest path
- Network length of longest path
- Path weight (sum of weights) on a weighted DAG, maximised or minimised
- Height and depth.
- Distance in antichains.

For instance in a DAG we could use HEIGHT DISTANCE $d(u, v) = (h_u - h_v)$ (if $u \succeq v$)

$$\sigma_{-v}^{(h)} = \sum_{u \in u \succ v} (h_v - h_u), \quad \sigma_{+v}^{(h)} = \sum_{u \in u \prec v} (h_v - h_u) \quad (3.3.4)$$

or indeed depth $d(u, v) = (d_v - d_u)$.

(T)We could look at the triangle identity deficit $d(f, i) - d(f, v) - d(v, i)$ which measure how far node v lies off the optimal path from i to f . Possible centrality measure. (end T)

⁴²For example for two points in two-dimensional space with coordinates (x, y) and $(0, 0)$ they are separated by distance $\sqrt{x^2 + y^2}$.

Pseudo-Distance, DAG Metrics and Geodesics

The key properties of the longest path can be summarised in a number of properties that we will encounter again, especially in terms of simple models based on a continuous space. For this reason, we now define what we will call a DAG METRIC^{43,44} where

$$d^{(\text{DAG})}(x, y) \geq 0 \text{ if } x \succeq y \quad \text{non-negativity,} \quad (3.3.5\text{a})$$

$$d^{(\text{DAG})}(x, x) = 0, \quad \text{reflexivity,} \quad (3.3.5\text{b})$$

$$d^{(\text{DAG})}(x, z) \geq d^{(\text{DAG})}(x, y) + d^{(\text{DAG})}(y, z) \text{ if } x \succeq y \succeq z \quad \text{reverse triangle identity.} \quad (3.3.5\text{c})$$

Note our notation follows the same convention as we use for the indices on the adjacency matrix. That is $d^{(\text{DAG})}(x, y)$ has a value when there is a path from y to x . The length of the longest path satisfies these conditions if we define a LONGEST-PATH METRIC $d^{(\text{lp})}(v, u)$ as follows

$$d^{(\text{lp})}(v, u) = d_{vu}^{(\text{lp})} = \begin{cases} d_{vu}^{(\text{lp})} & \text{if } v \succeq u \\ \text{undefined} & \text{otherwise} \end{cases} \quad (3.3.6)$$

We could use many labels other than undefined such as $\pm\infty$ but outside a numerical implementation it is better to think of this distance measure as only defined on connected node pairs. Note how the definition of $d^{(\text{lp})}(v, u)$ in terms of the longest path from u to v automatically gives us the reverse of the traditional triangle identity. That is no other path via an intermediary node w can be longer since by definition we defined the distance in terms of a longest path. We might also note that the longest-path metric satisfies a stronger condition that $d(x, y) = 0$ if and only if $x = y$ which is known as the identity of indiscernibles.

Finally we can define a geodesic in terms of this pseudo-distance measure $d^{(\text{DAG})}$. To do this we just have to ask that a geodesic is a path which is a local minimum *or* a local maximum. That is any small perturbation always makes a geodesic smaller, or it always make the path bigger. Adding this extra option makes no difference in the usual geometries we work in. For instance in Euclidean geometry, the geodesic is the (only) straight line between any two points and this is always the shortest path between those points. The triangle identity obeyed by the Euclidean metric ensures that the straight line is a local minimum. However, our DAG metric obeys the the reverse triangle identity so we need to extend our definition of a geodesic to allow for this case where we are looking for paths that are a local maximum in the length.

Network Length of Edges

(T)NO! Keep the network length of an edge to be one so the network length of a path is the number of edges i.e. sum of network lengths. Find another term for setting edge weights equal to the longest path distance. “longest path weight”??? (end T)

We will see that not all edges in a DAG play the same role. One way to reveal this is if we define NETWORK LENGTH of each edge (a type of edge weight) in a DAG to be equal to the longest path distance between the ends of that edge. That is for an edge (u, v) the network length of the edge is simply $d^{(\text{lp})}(v, u) = d_{uv}^{(\text{lp})}$. This is shown in Fig. 3.5.

The most important difference that this edge length reveals is between those edges of network length one and those edges with larger network lengths. If an edge (u, v) is has a network length of at least two then, by definition, there must be an alternative path linking the two ends of

⁴³Formally, this is a pseudo-metric and a pseudo-distance but for simplicity we will drop the ‘pseudo’ from the name.

⁴⁴It may sometimes convenient to define a SYMMETRIC DAG METRIC where we add the case that $d^{(\text{lp})}(v, u) = d_{uv}^{(\text{lp})}$ when $v \preceq u$ in (3.3.6). Then we would add the symmetry condition that $d(x, y) = d(y, x)$ for this symmetric version of the DAG metric.

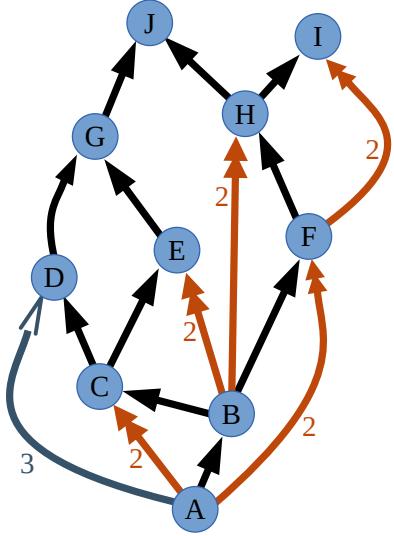


Figure 3.5: An illustration of the network length of an edge in a DAG using the same DAG as Fig. 2.6. The network length of an edge is equal to the longest path distance between the two nodes defining the edge. All the edges of network length 1 are shown as continuous straight black lines with no label and these are the essential edges. The remaining edges are the inessential edges labelled by their network length: five edges of network length 2 in red with double arrow head and the single edge (A, D) of network length 3 in blue.

Note that the two longest paths from A to J , (A, B, C, D, G, J) and (A, B, C, E, G, J) , only traverse essential edges of network length 1. On the other hand, the path (A, B, F, H, J) only traverses essential edges but this does not ensure it is one of the longest paths from A to J .

the edge, from u to v . Such an alternative path must not be direct, it must have at least one other node on it so such an alternative path must be of length two or more. In this case our direct edge (u, v) would carry a weight of two or more. The converse is that if there is no alternative route from u to v the edge (u, v) then we know that this edge is needed to maintain the connectivity of the graph. That is if we removed this edge from u to v , there would still be a path from u to v in the DAG without the direct edge. It that sense, the connectivity of the nodes in the DAG does not depend on the direct edge (u, v) . The converse is that if the network length of an edge is one, then there can be no alternative path from u to v . If there was, this would have to be longer and the network length of this edge would not be one. So we will call edges of network length one ESSENTIAL EDGES as they are required to maintain connectivity. Any edge of network length two or more will be called an INESSENTIAL EDGE.

We can go further and partition the set of edges into subsets of edges with the same network length. So $\mathcal{E} = \bigcup_{\ell=2}^H \mathcal{E}_\ell$ where \mathcal{E}_ℓ has edges of network length ℓ and the longest possible edge length is the height of the DAG H . So here \mathcal{E}_1 are the essential edges $\mathcal{E}^{(\text{ess})}$ while the rest are inessential $\mathcal{E}^{(\text{iness})} = \bigcup_{\ell=2}^H \mathcal{E}_\ell$. We will exploit the network lengths of edges in later sections.

3.4 Layering

(T)What sort of networks have the same longest and shortest paths between all nodes?
 Answer:- Transitively reduced graphs. For examples perfectly layered graphs where every edge is only connected to the next layer. Are these layered graphs always TR graphs? TR always removed shorter paths. However, there can be two paths from node u to node v where one path is not a subset of the other. Then these can be different lengths, e.g. Fig. 2.17 A to E .
(end T)

See Healy and Nikolov (2001) for clear definitions and classic methods. A LAYERING of a DAG is a partition of the nodes where the partitions form a sequence such that the nodes of one partition only point to nodes in partitions later in the sequence. RELATION TO ANTICHAINS? That is we have the partition of the set of nodes \mathcal{V} as $\mathcal{P} = \{\mathcal{V}_i\}$ where $\mathcal{V}_i \subseteq \mathcal{V}$, $\mathcal{V}_i \cap \mathcal{V}_j = \delta_{ij} \mathcal{V}_i$ and $\bigcup_i \mathcal{V}_i = \mathcal{V}$. We have a sequence as $i = 1, \dots, h$ where h is the HEIGHT OF THE LAYERING. Finally if $(u, v) \in \mathcal{E}$ is an edge and $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$ then $i < j$. Each partition \mathcal{V}_i is called a LAYER.

The HEIGHT OF THE LAYERING is the number of partitions. The WIDTH OF A LAYERwidth!layer is the number of nodes in a layer, $w_i = |\mathcal{V}_i|$. The WIDTH OF THE LAYERING is the largest of these widths, $w = \max(\{w_i\})$.

There are a number of possible layerings, as indicated in Healy and Nikolov, 2001 Healy and Nikolov (2001). The obvious one is to use what they call longest path. Each node is given a height which is the length of the longest of the longest paths to any sink node. Nodes of the same height are placed in the same layer and the height gives the sequence of the layers.

From Ugwu and Cuffe (2023)

This section introduces a modified version of the Horton-Strahler stream order numbering. The Horton-Strahler concept, named after the scientists Horton and Strahler who originally proposed it [22], establishes a statistical relationship between the river segments of different “orders” within a drainage basin, spanning from small originating streams which sequentially merge to form a large river mainstem. Stream order numbers quantify where in this branching hierarchy a particular river segment falls.

(T)TOTAL ORDER, TOTAL ORDER (end T) One of the properties of a poset is that we can always put the elements in what is called a TOTAL ORDER or TOTAL ORDER. This is a sequence $\mathcal{O} = (x_1, x_2, \dots, x_N)$ where $x_i \preceq x_j$ only if $i \leq j$ for $i, j \in \{1, 2, \dots, N\}$. The converse not true, so if $i < j$ this does not guarantee $x_i \preceq x_j$ but it does guarantee that $x_i \not\succ x_j$. Note that in general a poset or a DAG has more than one possible total order.

Chapter 4

DAG Models

- Set Inclusion
- Random Orders.
- Price model (BA model)
- Food Web models (interval graphs?)
- Poisson Point Process models/Box spaces:
 - Cube Space, Random orders
 - Minkowski (flat) space-time model, geodesics, comparison with shortest path and Euclidean space
 - de Sitter and curved space models
- Citation Models
- Configuration model? Or in numerical section.

4.1 Random Order DAG

Perhaps the simplest way to create a DAG is take any existing graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_G)$ and we assign an order to the nodes. In a RANDOM ORDER DAG \mathcal{D} the nodes are those in the original graph \mathcal{G} . However we now add distinct real number $t(u) \in \mathbb{R}$ as a label for each node $u \in \mathcal{V}$. These numbers will be chosen randomly drawing from the same distribution for each node. We then define directed edges (u, v) in the random order DAG to exist whenever the edge is in the original graph G , so $(u, v) \in \mathcal{E}_G$ but only if the label for the source node u is less than the label for the target node v , so $t(u) < t(v)$ is needed.

$$\mathcal{E} = \{(u, v) | (u, v) \in \mathcal{E}_G, t(u) < t(v)\}. \quad (4.1.1)$$

This will normally give a DAG that is neither transitively complete nor transitively reduced as illustrated in Fig. 4.1. This definition only relies on the order of the labels so this is equivalent to assigning the node labels from a random permutation of the numbers 1 to N , the number of nodes.

The classic example of this type of model is to start from an Erdős-Réyni random graph (see section 1.5.1). That is where we have a set of N nodes and we visit every pair of nodes and with probability p we connect that pair with an edge otherwise we leave this pair unconnected.

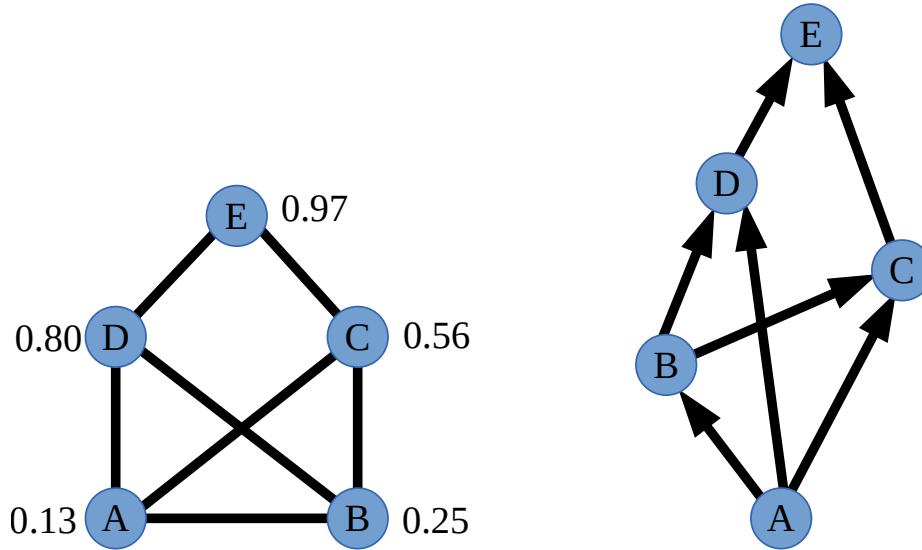


Figure 4.1: An example of the construction of a random order DAG. We start with the simple graph on the left. Each node is assigned a random number chosen uniformly at random from the interval $[0, 1]$ as shown to two decimal places beside each node. This then defines the DAG shown on the right where the node labels show where the nodes on the left have been drawn in their DAG equivalent on the right

(T)¹ To produce a DAG, we simply choose an order for the nodes and add a direction to the edges that respects that order. (T)²

We can view this Erdős-Réyni random DAG model in terms of an adjacency matrix. All we are doing is filling the entries of the lower triangular part of the adjacency matrix with 1's or 0's, each value chosen randomly. That is for edge (i, j) from node index i to node index j we set $A_{ji} = 0$ if $i \geq j$ and choose $A_{ji} = 1$ or 0 at random for $i > j$. One total order for this DAG comes from the integer index of the adjacency matrix.

Albert and Frieze 1989 look at one dimensional Random DAGs, looking at height (longest path) and width.(T)³

4.2 Set Inclusion

The SET INCLUSION DAG is defined in a simple but abstract way. While it is unlikely that a model of a DAG will be defined directly in this format there are many models which can be reinterpreted in this language as we see later.

The set inclusion DAG is defined as follows. Suppose we have a collection of N sets $\{S_u\}$ where each node u in a DAG represents one of the sets S_u in this collection. These sets are distinct so that the elements of the set S_u linked to node u are exactly the same as those in any other set S_v linked to a different node v . There is a directed edge from node u to node v provided the set linked to node u is a subset of the set associated with node v , that is $(u, v) \in \mathcal{E}$ if and only if $S_u \subset S_v$. See Fig. 4.2 for an example.

¹(T)This is a convenient definition mathematically but numerically, it is usually simpler to pick fix the number of edges to be E . Then we pick one pair of nodes uniformly at random from the set of all possible $N(N - 1)/2$ pairs and we add a connection provided no edge is already present. We repeat this until E edges are attached. If we choose $E = p(N - 1)$ then the sets of random graphs produced by these two approaches are very similar, becoming more and more similar as the number of nodes N grows.

²(T)Given the edges were chosen at random, it doesn't really matter what order we give to the nodes.

³(T)Add info?

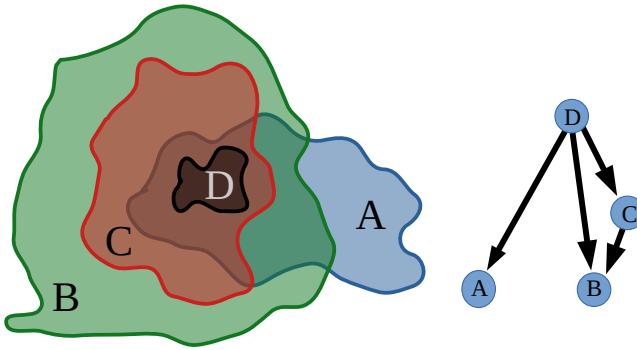


Figure 4.2: An example of a set inclusion DAG. On the left the Venn diagram represents the intersection of four sets A, B, C and D . Using a set inclusion rule gives the transitively complete DAG on the right. All the edges, both solid and dashed, are present in the DAG which is transitively complete. However, we would normally just use the Hasse diagram visualisation where the dashed lines would be left out and just the transitively reduced graph would be shown.

Note that this definition always gives a transitively complete DAG. Put another way, set inclusion always defines a partially ordered set. That is we define the partial order such that $u \preceq v$ if $S_u \subseteq S_v$. This means that reflexivity $u \preceq u$ is satisfied since formally $S_u \subseteq S_u$. The transitivity condition needed for the poset (2.9.3) follows naturally. If $S_u \subseteq S_v$, and $S_v \subseteq S_w$, we know that set S_u is also a subset of S_w . In the poset language we write this as $u \preceq v$ and $v \preceq w$ and this then always implies $u \preceq w$ i.e. transitivity (2.9.3). In terms of the edges in a DAG, these conditions mean that any set inclusion DAG is always a transitively complete DAG.

4.2.1 Geometric Set Inclusion

One place where set inclusion DAGs are likely to be encountered is in a geometric context. In this case the sets $\{S_u\}$ are all closed volumes in a d -dimensional space. Each closed volume S_u defines a node u in our DAG. Then we use the set inclusion rules to define the edges. That is if one of these closed volumes S_u is completely enclosed in another S_v , so $S_u \subset S_v$, then we say that there is an edge from u to v . In fact the Venn diagram representation in Fig. 4.2 already shows how this works.

One of the most common examples would be spheres or balls in some metric space, a set of points \mathcal{X} and a distance $d(\mathbf{x}, \mathbf{y})$ between any points $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. The simplest example would be D -dimensional Euclidean space. Normally, our nodes \mathcal{V} would be a given subset of points within the space, so $\mathcal{V} \subseteq \mathcal{X}$. We then define closed volumes of radius r_u around each node u in our network; this would be the region $S_u = \{d(\mathbf{x}, \mathbf{u}) < r_u\}$ so that node u in the DAG is a hypersphere centred on \mathbf{u} and of radius r_u . Then use set inclusion to define the edges based on the set of points S_u linked to each node, so edges (u, v) exists if $S_u \subseteq S_v$. We show this in two-dimensional Euclidean space with circular disks in Fig. 4.3.

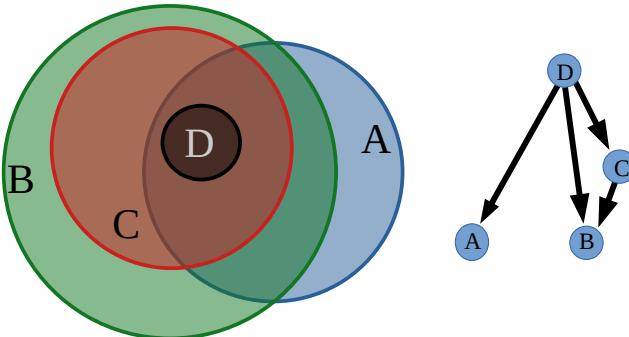


Figure 4.3: An example of a set inclusion DAG given in terms of circles. Using a set inclusion rule on the four circles A, B, C and D gives the transitively complete DAG on the right.

Another example of a geometric set inclusion model which we can interpret as a transitively complete DAG comes from DISK SPACE, see (Suzuki et al., 2019) and references therein. The idea is that we define a standard shape, our DISK $\Delta(\mathbf{x} = 0, r = 1)$, in some d -dimensional

vector space, say X . These can be scaled by a factor r and moved to be centred on any point $\mathbf{x} \in X$ to give a disk $\Delta(\mathbf{x}, r) \subset X$. Each disk represents one node in our DAG. We then use set inclusion on the disks to define the edges in our DISK SPACE DAG .

Typically disks are polyhedra our d -dimensional space X where the disks are specified by a set of at least $(d + 1)$ fixed vectors $\{\mathbf{w}^{(i)}\}$. Geometrically, for any disk $\Delta(\mathbf{x}, r)$, for each vector $\mathbf{w}^{(i)}$, we place a $(d - 1)$ -dimensional hypersurface which is perpendicular to $\mathbf{w}^{(i)}$ and which passes through the point $\mathbf{x} + r\mathbf{w}^{(i)}$. These hypersurfaces are then the boundaries for a hypervolume containing the point \mathbf{x} and this is the disk $\mathbf{x} + r\mathbf{w}^{(i)}$. This is illustrated in Fig. 4.4. Another example is that we can choose our disks to be hypercubes as follows. If $\mathbf{e}^{(i)}$ are

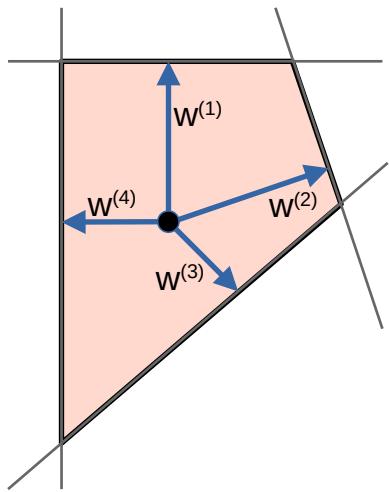


Figure 4.4: An example of a ‘disk’ used to define disk space. Each disk is defined by points, the centre of the disk shown here as a black circle, a set of at least $d + 1$ vectors in a d -dimensional Euclidean space. Here, we have four vectors in two dimensions: $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)}$, and $\mathbf{w}^{(4)}$ shown here as blue arrows. These vectors point out from the centre of the disk. Each vector defines a $(d - 1)$ -dimensional hypersurface at the end of the vector. Here these hypersurfaces are the one-dimensional lines that are orthogonal to the vectors and which pass through the point of the arrow. The d -dimensional volume contained within the intersection of these hypersurfaces and which contains the central point, is the ‘disk’, here the two-dimensional irregular four-sided polygon, the area shaded red.

basis vectors for the vector space X then we define $2d$ vectors $\mathbf{w}^{(2i-1)} = +\mathbf{e}^{(i)}$ and $\mathbf{w}^{(2i)} = -\mathbf{e}^{(i)}$ for $i = 1, 2, \dots, d$.

We see further examples of DAG models that can be interpreted from this geometric set inclusion viewpoint later.

(3D geometric picture of cones)

Figure 4.5: An example of a set inclusion DAG created in a geometric context. Try to get this to reproduce the DAG example in Fig. 4.3.

4.3 The Poisson Point Process for nodes

- Cube Space
- Minkowski (flat) space-time model, geodesics, comparison with shortest path and Euclidean space
- de Sitter and curved space models

We have already seen in section 4.2.1 how we can use geometry to create DAGs when we looked at examples of the set inclusion definition using areas/volumes based around a point in some given space. There are many ways to give coordinates to nodes so that we have a DAG embedded in some space. Sometimes there are natural coordinates for nodes in a given data set. In this section, we look at one standard way to give coordinates to nodes for use in a

variety of models. In the following sections we look at different models which define the edges between nodes that have some coordinates.

In the **POISSON POINT PROCESS (PPP)** points are thrown down randomly in some space but each point is placed with exactly the same probability independent of any points already placed. More precisely, we assume the number of points found in any region R follows a Poisson distribution⁴ (see appendix A.4.3). All we need to specify is the average number of points N_R we will find in any given region R within our space⁵.

We will focus on the simplest example. In our PPP models, we will assume that the nodes are embedded in a D -dimensional space so that each point x is specified by D real coordinates $x_i \in \mathbb{R}$, $i = 1, 2, \dots, D$. We will also assume that the density of points ρ is constant so that the average number of points N_R in any region R of volume μ_R is specified by this single parameter ρ with $N_R = \rho\mu_R$ for any region R .

(T)⁶

In practice the simplest way to implement this model numerically is to throw down exactly N points in a specified region of volume V so our constant density is $\rho = N/V$. If N is a fixed number, then we actually have a binomial distribution for the number of points in any given region of volume μ with the mean number of points $(\mu/V)N$. For large numbers, there is little difference between the true Poisson point process and this binomial equivalent but in practice, the Poisson viewpoint is usually better for mathematical analysis while the binomial version is best for numerical work⁸.

To throw the points down, the simplest approach is to choose the coordinate of each point uniformly at random from some range, say 0 to L_i for which coordinate i . That way our points lie in a hyper-rectangular space of size $L_1 \times L_2 \times \dots \times L_D$. If we wanted our points to lie within a region which was not hyper-rectangular, one approach is to choose the hyper-rectangular region to be big enough such that our desired region is contained within. We just ignore any points that do not lie in the desired region and we continue to add more points until we have enough in our desired region⁹.

To illustrate the PPP models let us consider a specific example. We will work with N points, each with D real coordinates each lying between $[0, 1]$. For each point, we choose each of the D coordinates chosen uniformly at random from $[0, 1]$. Assuming that our volume elements ΔV are the simple infinitesimal elements $dV = \prod_{i=1}^D dx_i$ then our total volume is 1 and our constant density is simply $\rho = 1/N$.

The PPP process defines the positions of the nodes in our space. We now need to consider the rules needed to connect them, the definitions of the edges. There are several ways to do this and each leads to a different type of model for a DAG.

⁴One way to visualise this is to imagine that we have a large but finite number of points $N^{(\text{tot})}$. Each point has a finite probability p of being placed in some given region R of volume μ_R . We take $N^{(\text{tot})}$ to be large but we set p by specifying that the average number of points in our region R is equal to some specified number N_R . While formally the distribution is a binomial, in the large $N^{(\text{tot})}$ limit we have the Poisson distribution where we choose p such that $pN = N_R$.

⁵Formally, if we want a set of volume elements $\{\mu_i\}$ which do not overlap, so $\mu_i \cap \mu_j = \emptyset$ if $i \neq j$, but which cover the total volume V , that is $\cup_i \mu_i = V$. Even more formally, this means the least geometric structure our space must have is that it is a MEASURE SPACE. We will not pursue this general measure space viewpoint here but see Bollobás and Brightwell (1991) as an example where this formal language is used.

⁶(T)(LEAVE OUT?) The name Poisson point process comes from a viewpoint where the points placed in the volume V with a certain probability per unit time τ , where this is ‘model’ time⁷, and this is done over a given time T . This means the total number of points N is not fixed for any one instance but the mean number expected in the total volume $\langle N \rangle$ is fixed.

⁸If you wish, instead of specifying a fixed N , we could specify a parameter equal to the average number of points $\langle N \rangle$ expected in our total volume. We would then draw the actual number of points N to put down in our volume from a Poisson distribution with this mean, using a suitable library function to do this.

⁹This accept-reject approach is simple but can be inefficient. If we know the shape of the region we want, there maybe more efficient ways to generate points.

4.4 Cube-Space DAG

In a CUBE-SPACE DAG, each node v also carries D real coordinates $\{v_1, v_2, \dots, v_D\}$. In our context, these come from the PPP process but the coordinates could have other origins especially when dealing with real data, for instance see the ranking and rating DAGs of section 5.4. The edges in a cube-space DAG an edge from node u to node v exists if and only if every coordinate of u is less than the corresponding coordinate of v . This is the CUBE-SPACE RULE which is given formally as

$$(u, v) \in \mathcal{E} \quad \text{iff} \quad u_i < v_i \quad \forall i \in \{1, 2, \dots, D\}. \quad (4.4.1)$$

The reason for calling this rule a cube-space rule is that we can picture the cube-space DAGs as being embedded in a D -dimensional space where the u_i are coordinates of the node along orthogonal axes. The only possible edges from u are to nodes v which lie in a D -dimensional hypercube where u is at one corner as shown in Fig. 4.6.

It is important to note that the definition of a cube space DAG (4.4.1) does not involve any sense of distance between points. All that is required is the natural sense of order inherent in the coordinates v_i so we can make the comparison of coordinates in the $u_i < v_i$ criteria of the cube space rule (4.4.1).

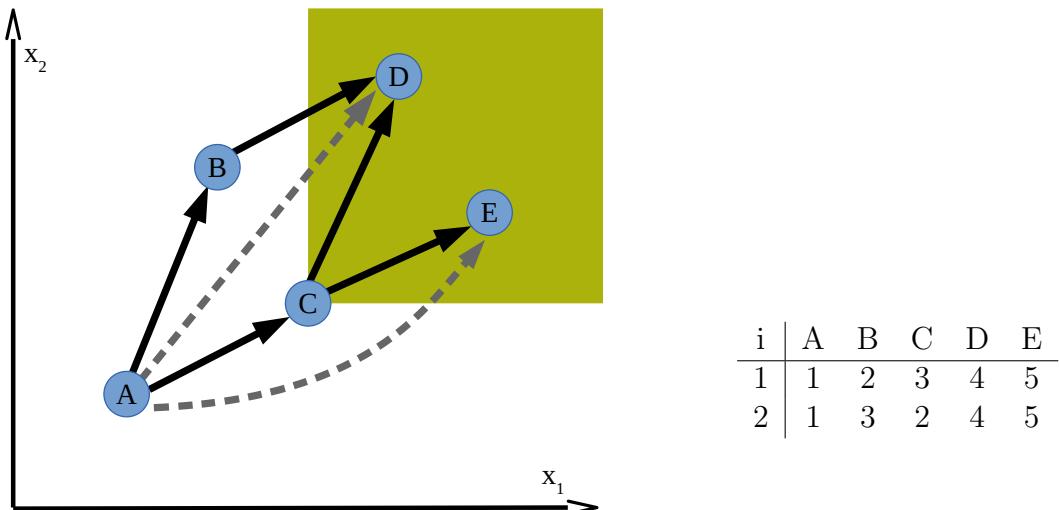


Figure 4.6: An example of a cube-space DAG shown with its natural embedding into two-dimensional space \mathcal{X} . The square with node C at the lower left corner is the forward-connection kernel $\mathcal{N}^+(C)$ for node C and any points in that area will receive an edge from node C . All the edges, both solid and dashed, are present in the DAG which is transitively complete. However, we would normally just use the Hasse diagram visualisation where the dashed lines would be left out and just the transitively reduced graph would be shown. On the right the ranks of each coordinates is shown for each of the dimensions $i = 1, 2$. Each row is a permutation of 1 to 5. We could use these ranks as coordinates and we would produce the same DAG as the cube space rule (4.4.1) relies on an order in the coordinates and not any sense of distance between points in the embedding space \mathcal{X} .

We will find that in many models it is useful to think about the FORWARD CONNECTION KERNEL $\mathcal{N}^+(x)$, the set of points that are possible targets for edges starting from one given point x . These are the hypercubes that give the cube space model its name. Here we have that

$$\mathcal{N}^+(x) = \{y \mid y \in \mathcal{X}, \quad x_i < y_i \quad \forall i \in \{1, 2, \dots, D\}\} \quad (4.4.2)$$

with an example of the forward connection kernel in a cube space model shown in Fig. 4.6. There is a similar BACKWARDS CONNECTION KERNEL $\mathcal{N}^-(x, R)$, the predecessors of x ,

$$\mathcal{N}^-(x) = \{y \mid y \in \mathcal{X}, \quad x_i > y_i \quad \forall i \in \{1, 2, \dots, D\}\}. \quad (4.4.3)$$

Again this is a hypercube in the cube space model. Note that these definitions of forwards and backwards connection kernels exclude the source/target node x of the connection kernel but we could define include this point versions to defined closed versions $\widehat{\mathcal{N}}^+(x)$ and $\widehat{\mathcal{N}}^-(x)$ replacing $<$ by \leq in (4.4.2) or $>$ by \geq in (4.4.3) so the point x is included.

This geometric picture of the cube space model in terms of these forward connection kernel hypercubes also allows us to reinterpret this model as a type of set inclusion model of section 4.2.1. That is if there is an edge from u to v then the backwards connection kernel of u is completely contained within the backwards connection kernel¹⁰ of v , so $(u, v) \in \mathcal{E}$ iff and only if $\mathcal{N}^-(u) \subset \mathcal{N}^-(v)$.

The cube space rule alone will always give a transitively complete DAG. That is the edges in cube space DAG are representing nodes which are related by a binary relation in the corresponding poset. We define the required binary relation \preceq as

$$u \preceq v \Leftrightarrow (u, v) \in \mathcal{E} \text{ or } u = v \Leftrightarrow u_i \leq v_i \forall i \in \{1, 2, \dots, D\}. \quad (4.4.4)$$

We can see that transitivity is obeyed by this relation as follows. Suppose that $u \preceq v$ and $v \preceq w$. The cube space rules used to define these edges means that for each dimension i we have that $u_i \leq v_i \leq w_i$. Therefore $u_i \leq w_i$ so $u \preceq w$ and either $u = w$ or there is also an edge from u to w . This gives us the transitivity requirement for (\mathcal{X}, \preceq) to be a poset and it is straightforward to confirm that the other criteria for a poset hold. The transitivity requirement also shows that if there is a long path between two nodes, we can take any three consecutive nodes in the path and cutout the middle node to leave us with another valid path. Repeating this process eventually shows there is a direct edge between any two distinct nodes that are connected by a non-trivial path and this defines a transitively complete DAG.

The fact that we only need a sense of order for each dimension separately also tells us that we can work in terms of the rank of each node for a given dimension. That is the i -th rank coordinate for node v is an integer $v_i^{(\text{rank})}$ between 1 and N which is equal to the number of nodes u whose i -th coordinate u_i is less than or equal to $v_i^{(\text{rank})}$. That is $v_i^{(\text{rank})} = |\{u | u_i \leq v_i | u \in \mathcal{V}\}|$. Now we see that the cube rule applied to these rank coordinates gives the same result as when using the original values, that is $v_i^{(\text{rank})} < u_i^{(\text{rank})}$ iff $v_i < u_i$ since we only used the order of the coordinates in the cube space rule.

Working with the ranked values of the coordinates set by PPP is equivalent to working with D randomly chosen permutations of the integers from 1 to N . This is view of the model is what is called a ‘random order’ in mathematics and indeed the cube space rules emerged first from this context (Winkler, 1985a,b).

The interpretation of the coordinates in the cube space DAG as D permutations of $(n)_{n=1}^N$ gives some quick insights into the model. First, the values of the coordinates v_i , or their ranked values v'_i , of any one of the dimensions i gives a total order. The definition of edges in the cube space DAG (4.4.1) in terms of the coordinate u_i for any one dimension i is consistent with the $t(u)$ values used in the definition of a total order in section 2.6. That is we can choose $t(u) = v_i$ (or v'_i) for any fixed i and we will always have that $t(u) < t(v)$ for any edge (u, v) in the DAG. It means that we can always relabel our nodes and choose a new index for each node such that for one of the dimensions, the first node has the lowest v_i , the second node has the next lowest value and so forth.

It is simple to add additional rules which destroy transitivity and to produce a full range of DAGs from this model. For instance we could combine the cube space rule with the random order DAG to give a D -dimensional random order DAG. That is rather than having one total

¹⁰We could use the forwards connection kernel, perhaps that feels more natural. In this case if $(u, v) \in \mathcal{E}$ we have that $\mathcal{N}^+(v) \subset \mathcal{N}^+(u)$ which is the reverse of the definition we used for our set inclusion DAG of section 4.2.1. However, reversing the edges, i.e. the direction of the order in the DAG, in any of these definitions is a trivial symmetry of any DAG. So it is always a matter of preference and of no mathematical consequence.

order added to nodes in an existing network \mathcal{G} we now think of adding D coordinates to each node. We then demand that edges are present in the DAG only if they exist in the original graph \mathcal{G} and if the coordinates satisfy the cube space rule (4.4.1)

$$\mathcal{E} = \{(u, v) | (u, v) \in \mathcal{E}_G, u_i < v_i \quad \forall i \in \{1, 2, \dots, D\}\} \dots \quad (4.4.5)$$

Alternatively we could view this as adding some extra test to the cube space rules (4.4.1) when creating edges such as only adding edges with some probability p when they obey the cube space rule. A simple case such as this is equivalent to combining an Erdős-Réyni graph (see section 1.5.1) with the cube space rules. Another example would be where we impose some cutoff on the length of edges added which means we need to have a sense of distance between points and not just a sense of order in each coordinate. This latter approach is equivalent to setting the original graph \mathcal{G} to be what is called a random geometric graph and we study this case below in section 4.3.

4.5 The Random Geometric Graph model

The next few models use a geometric approach to create a DAG. We have seen how a Poisson point process can be used to produce random points in a space for a model. In a geometric approach to DAG model making we then need a measure of distance between points in that space to use when creating the edges in the model.

We will start by looking at a standard model that uses geometry to produce an undirected network, known as the RANDOM GEOMETRIC GRAPH model (**RGG**). Typically the N nodes in the model are given coordinates in a D -dimensional space by using the Poisson point process of section 4.3. To add edges, we need a sense of distance $d(x, y)$ between any two points x and y . This is usually a distance in the strict sense of mathematics as discussed in section 3.3, that is the distance function is never negative $d(x, y) \geq 0$, it is symmetric $d(x, y) = d(y, x)$ (the distance from x to y is the same as the distance from y to x) and the distance between two points is zero between a point and itself $d(x, x) = 0$ (and is usually non-zero otherwise). Many distance measures also obey the triangle inequality triangle inequality $d(z, x) \leq d(z, y) + d(y, x)$ (3.3.2) in which case the distance is called a metric. The typical example we will use is the Euclidean metric $d(x, y) = [\sum_{i=1}^D (x_i - y_i)^2]^{1/2}$.

The standard RGG starts with N nodes as points assigned using the Poisson point process in a D -dimensional space. These nodes are then connected if they are within a distance R from each other so this RGG is a one-parameter model¹¹. Properties of the resulting network will only depend on the average number of points within a circle of radius R and we can either change the density of points or R to change this (Penrose, 2003, Ellis et al., 2007, Díaz et al., 2016, Barthelemy, 2018, Kartun-Giles et al., 2019). Note there is no sense of direction in this model¹² so edges are undirected and this basic RGG model gives a simple network.

We saw in section 4.1 that a simple way to make a DAG from an existing network is to just add a total order to the nodes. So consider what we call a ORDERED RANDOM GEOMETRIC GRAPH (**ORGG**) model . In principle this can be a space of a finite number of discrete points but we will assume it is a continuous space of points in a D -dimensional space. That is points $\mathbf{x} \in \mathcal{X}$ can be described by D real coordinates x_i . The typical example is Euclidean space $\mathcal{X} = \mathbb{R}^D$ where the distance between points is given by the usual Euclidean measure of

¹¹There is another length scale which is the distance between points which is proportional to $\rho^{-1/D}$. Formally it is the ratio of these two length scales which is the length parameter in this model. This also assumes we are working in a very large space otherwise the size of the region containing points will set another length scale which can effect results.

¹²Except for the effect of the boundaries. The properties of the model will change close to boundaries of the region used in the model.

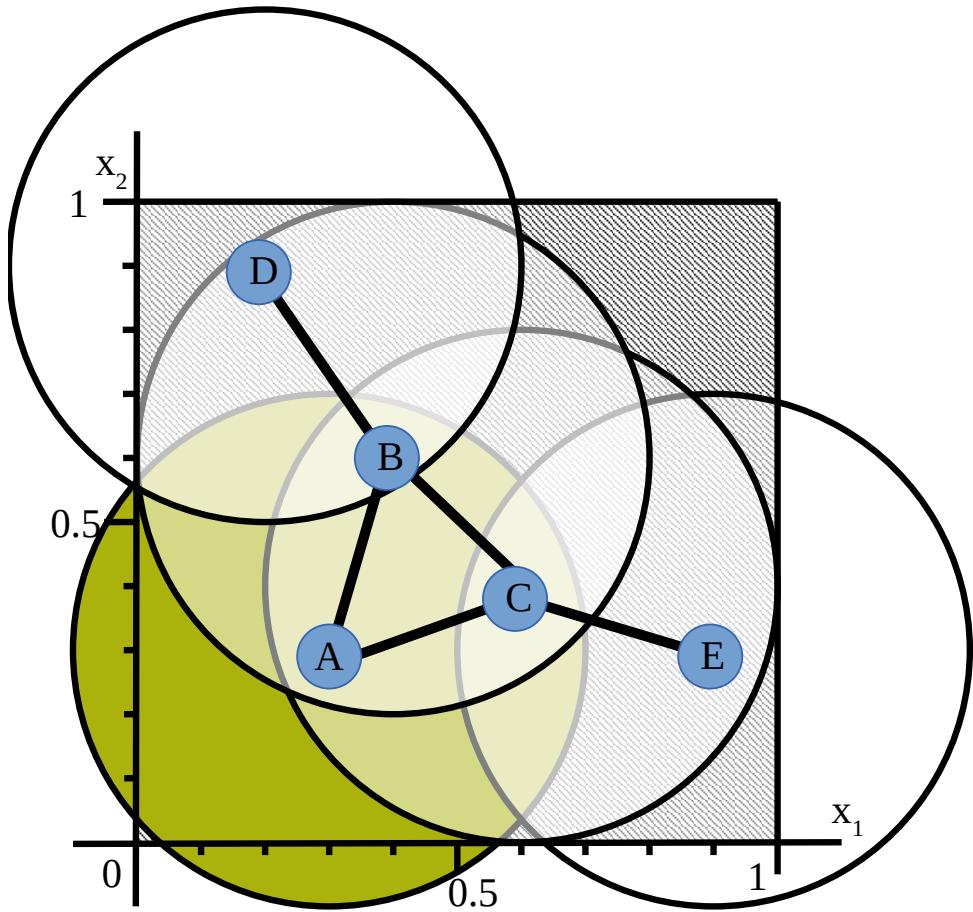


Figure 4.7: An example of a random geometric graph. Here five points are created where the coordinates (x_1, x_2) of each point \mathbf{x} are chosen uniformly at random to lie between 0 and 1. The range of locations of points is shown by the shaded background. These points are the five nodes of this network. In this case edges are chosen by connecting any point \mathbf{x} to all points within a radius R as measured by a Euclidean metric. That is the connection kernel is a circle of radius R centred on the point \mathbf{x} . This is shown for point A as a filled green circle while for other points only the perimeter is shown. This results in the addition of five undirected edges as shown by the solid black lines. This basic RGG model gives a simple network.

distance. So to make this a DAG using the random order graph scheme we add a total order. In this case we will add a ‘time’ coordinate $t \in \mathbb{R}$ to each point \mathbf{x} . So now our nodes are embedded in a ‘space-time’ $\mathbb{R} \times \mathcal{X}$ with coordinates $(t, x_1, x_2, \dots, x_D)$.¹³

So formally the ordered RGG is represented by the DAG $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ with node set \mathcal{V} , each node u with a time coordinate $t(u)$ and spatial coordinates that allow us to calculate the distance between node pairs. Then the set of directed edges \mathcal{E} is given by

$$(u, v) \in \mathcal{E} \quad \text{iff} \quad t(u) < t(v) \quad \text{and} \quad d(v, u) < R. \quad (4.5.1)$$

A simple example of a DAG from an ordered RGG model is shown in Fig. 4.8.

One special case of this DRGG model is where we take $R \rightarrow \infty$ so that the distance function plays no role. In the standard Euclidean RGG this would mean every point is connected to every other point, a model of no particular interest. Here, however, the partial order alone

¹³We could take this further and suggest that we only need a partial order not a total order. That is we need a poset relation \preceq , and a distance function d defined on points related by the posets. We will not follow this option further.

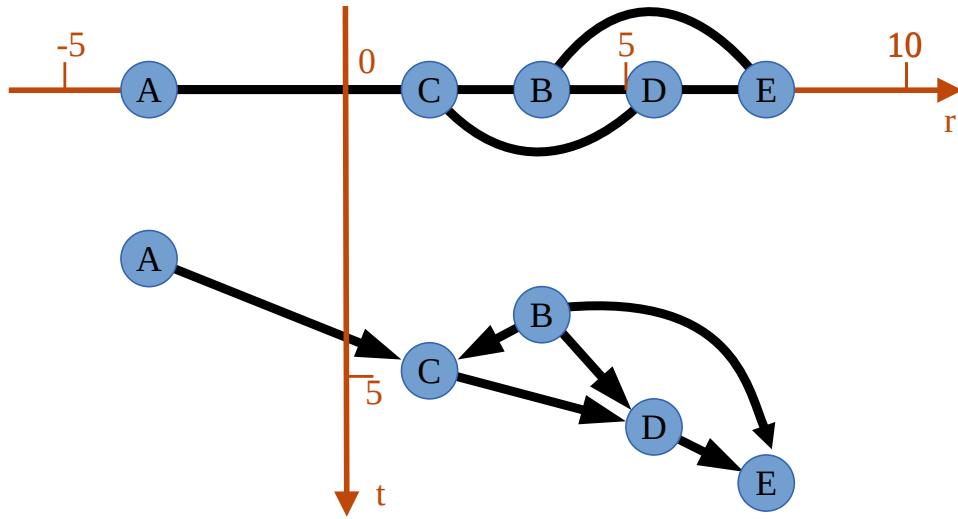


Figure 4.8: An example of an ordered random geometric graph. Here five nodes in a RGG are embedded in a one-dimensional space as indicated by the horizontal axis labelled r . These nodes are connected if the distance between the points (the difference in their r coordinates) is less than $R = 5.5$. The resulting undirected edges are shown in the simple network at the top (along the $t = 0$ line). To create a directed RGG we follow the prescription of the random order DAG and we add a time coordinate to each node. This is indicated by the vertical axis, the t values. The edges in the original RGG are preserved but are now given a direction from smaller to larger t coordinate. The results is a DAG shown in the lower part of the figure.

can and does lead to a non-trivial DAG. By definition it must be transitively complete as connections are derived solely from the properties of the poset.

We also note that we could take this further and apply a similar cutoff in the time coordinate, that is we could require that $0 < t(v) - t(u) \leq T$ where T is some time scale parameter. The case we gave above is the limiting case where $T \rightarrow \infty$. However, what we have done is patched on a one-dimensional time coordinate to an existing network model embedded in some D -dimensional space. We can imagine many other ways to impose suitable time and space constraints to get a DAG in these types of ordered RGG models. We will not pursue this approach further here but will instead move onto to look at some interesting examples of DAG models where we have time and space dimensions mixed together in a more coherent manner, i.e. we will look at models for DAGs embedded in a space-time.

4.6 Minkowski Space-Time DAG

The MINKOWSKI SPACE-TIME DAG exploits the properties of the simplest space-time as used in special relativity. The same definition holds for other space-times, such as those encountered in general relativity and in cosmological applications, but we will not consider these generalisations explicitly.

In a D -dimensional Minkowski space-time, denoted as \mathbb{M}^D , each point x has D -coordinates where one coordinate plays a special role, that of the time coordinate. We will follow standard relativistic notation and give each point x coordinates indexed from 0 to $(D - 1)$ where x_0 is the time-coordinate and x_1 to x_{D-1} are the spatial coordinates. In Minkowski space-time there is a natural measure of the separation between two points $x, y \in \mathbb{M}^D$ known as the PROPER

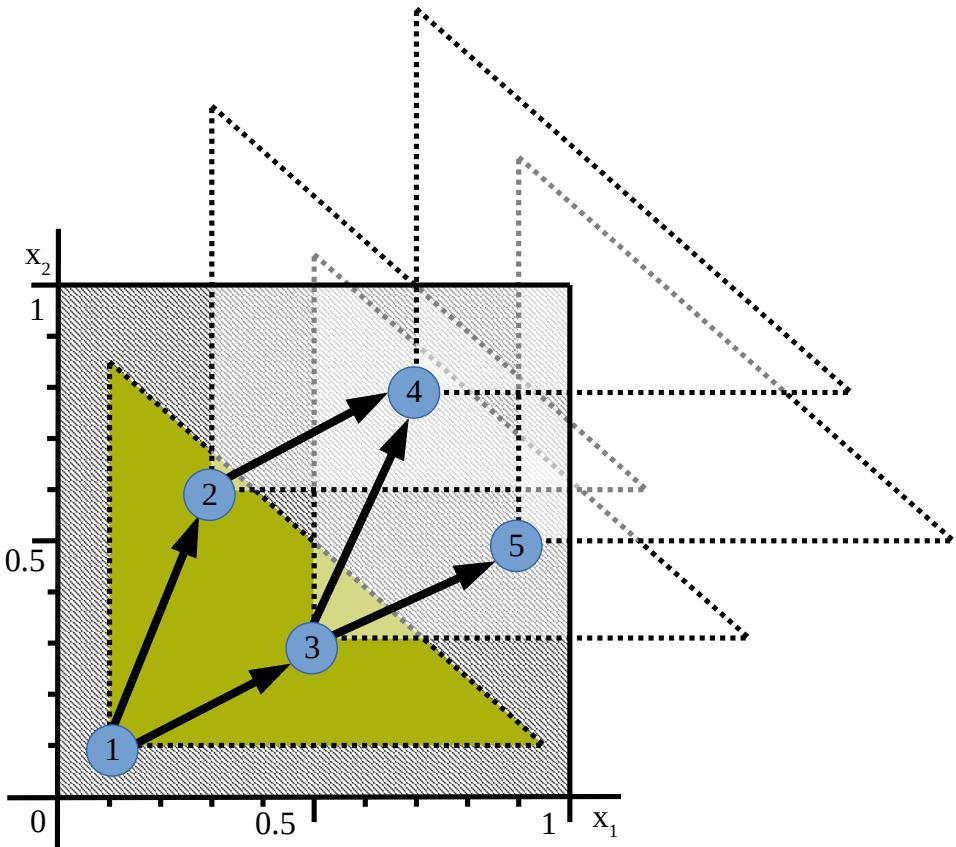


Figure 4.9: An example of a directed random geometric graph. Here five points are created where the coordinates (x_1, x_2) of each point \mathbf{x} are chosen uniformly at random to lie between 0 and 1. The range of locations of points is shown by the shaded background. These points are the five nodes of this network. In this case edges are chosen by defining the forward connection kernel to be a right-angled triangle where the right-angle is centred on each point and the short sides are equal length and parallel to axes. The forward connection kernel for point 1 is shown with a filled green triangle while for other points just the perimeter is shown by dotted lines. This results in the addition of five directed edges as shown by the solid black lines.

TIME τ and this is defined to be

$$\tau(\mathbf{x}, \mathbf{y}) = \left((x_0 - y_0)^2 - \sum_{i=1}^{D-1} (x_i - y_i)^2 \right)^{1/2}. \quad (4.6.1)$$

One way to understand where this strange looking ‘distance’ measure comes from is by looking at the symmetry of space-time. In particular when we change the coordinate systems, for instance look at distances as seen by moving observers, they all agree that the proper-time separation between two space-time points (‘events’) has the same value. It is important to remember that many DAG contexts are not underpinned by such symmetries but we can still see if this space-time model is a useful tool.

We will take the positive value from the square root when the argument of the square root is positive. However, the argument of the square root is negative whenever $|x_0 - y_0| < r$ where $r = \sqrt{\sum_{i=1}^{D-1} (x_i - y_i)^2}$ and then formally τ is imaginary. For instance, if our two points have the same time coordinate, $x_0 = y_0$ but are separated in space, e.g. $x_i \neq y_j$ for all $i = 1, 2, \dots, (D-1)$. For this reason, τ is *not* a formal distance function, as defined in (3.3.1), so it is some sort of pseudo-distance function.

The pseudo-distance function $d^{(\tau)}$ we will use in our Minkowski space-time DAG will avoid these imaginary proper time τ values as we define

$$d^{(\tau)}(\mathbf{x}, \mathbf{y}) = \begin{cases} \left| (t^2 - r^2)^{1/2} \right| & \text{if } t^2 \geq r^2 \\ \text{undefined} & \text{if } t^2 < r^2 \end{cases} \quad \text{where } t = (x_0 - y_0), \quad r^2 = \sum_{i=1}^{D-1} (x_i - y_i)^2. \quad (4.6.2)$$

We could use many labels other than undefined, $\pm\infty$ or any imaginary number are obvious choices. This value will never be used in our model because these are explicitly excluded by our definition of the partial order which is

$$x \preceq y \Leftrightarrow t^2 \geq r^2 \quad \text{and} \quad x_0 \leq y_0, \quad (4.6.3)$$

where t and r are defined as before in (4.6.2).

Another way to view this Minkowski space-time DAG model, one which will connect with the pyramid and disk models of section 4.7, is to see this as a geometric model. Essentially the forward and backwards connection kernels $\mathcal{N}^\pm(\mathbf{x})$ of a point \mathbf{x} are cones when $D = 3$ whose apex is at the point \mathbf{x} and where the symmetry axis of the cone is the time axis. If we take a slice through these cones at constant time they are circles when $D = 3$ (and hyperspheres in general) but circles whose radius is proportional to the time difference between the time where the slice is taken and the time coordinate of the apex x_0 . This is shown in Fig. 4.10.

In terms of a model for a DAG, we can simply take these abstract or geometric definitions as given. The nodes can be a set of points at chosen at random. The Poisson point process is the simplest to implement as we define a constant density of points in terms of the volume elements of the Minkowski space-time. In terms of the coordinate system used above this simply means that for each node v we choose the individual coordinates, both time and space coordinates, uniformly at random from a given range.

If we put no constraint on the distance between points, essentially the $R \rightarrow \infty$ limit of this Minkowski space-time RGG model, then the partial order constraint (4.6.3) produces a transitively complete model. The causality constraint $|t| > |r|$, which defines the partial order of this directed RGG, ensures many points are not connected giving us an interesting model though one that is completely defined by a poset. This Minkowski space-time RGG model has been studied both analytically, as the CONE-SPACE MODEL of Bollobás and Brightwell (1991), and numerically (Ilie et al., 2006, Rideout and Wallden, 2009, Clough et al., 2014, Clough and Evans, 2014, 2016, Vasiliauskaitė and Evans, 2020).

One motivation for these choices comes from physics of special relativity. The partial order constraint (4.6.3) is saying that we will only connect points in our DAG when the points are causally connected, that is when it is possible for information to travel from x to y , at the speed of light¹⁴ or less. For such points, order defined in the partial order reflects the arrow of time, with the direction of edges going from the earlier points in time to the later points in times. The properties of Minkowski space time ensure there are no loops as this reflects causality in the real world, the future cannot influence the past. The physical causality properties of this model also ensure that the DAGs produced are always transitively complete.

The use of Minkowski space-time geometry in special relativity leads to some specific nomenclature in this model. For instance, the future set of any point has the shape of a cone in the x_μ ($\mu = 0, \dots, (D-1)$) coordinates used here and this is known as the ‘forward light-cone’ in relativity. This is also why the model is sometimes referred to as cone space (Bollobás and Brightwell, 1991).

¹⁴Note that we are choosing units where the speed of light is equal to 1.0 in these units. For instance the time could be measured in seconds while distances are measured in light seconds, the distance travelled by light in one second.

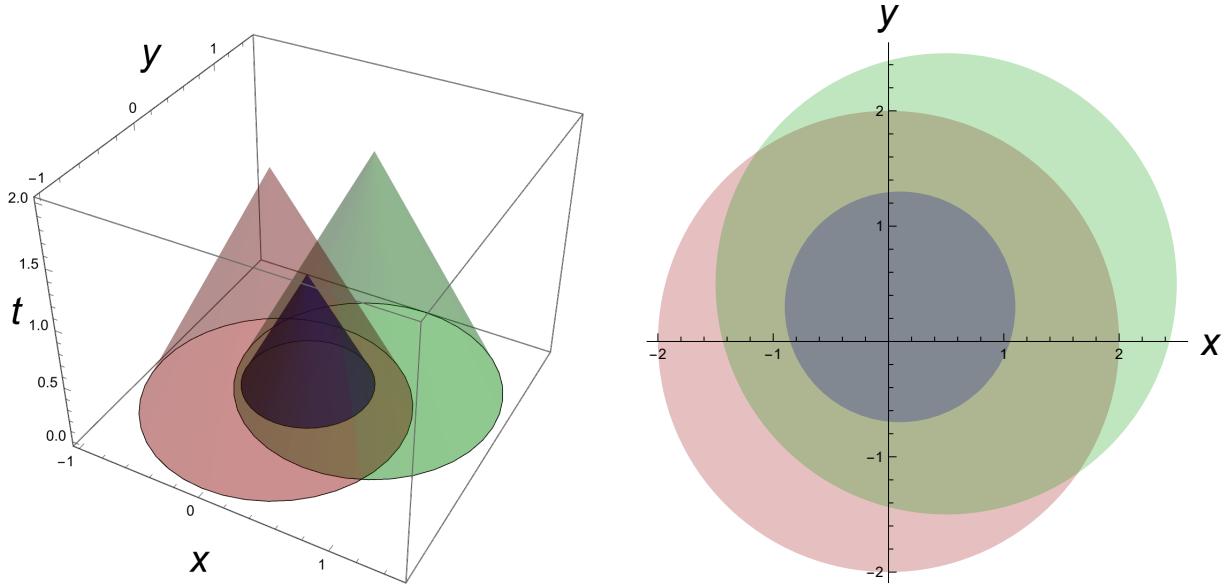


Figure 4.10: This shows how the backwards connection kernels of three points in a Minkowski space-time are used to define edges. The apex of each cone gives the coordinates of three nodes in this DAG at $(t, x, y) = (2.0, 0, 0, 0, 0)$ (red), $(2.0, 0.5, 0.5)$ (green) and $(1.0, 0.1, 0.3)$ (blue). The boundaries of this cone show that $c = 0.5$ in these units. We can use set inclusion of section 4.2 to define the DAG so that there is an edge from the node at the apex of the smaller blue cone to the nodes at the apex of other two cones since the smaller blue cone is completely within the other two cones. Equally there is no edge between the nodes at the apex of the two larger cones (red and green) since neither of these cones is completely contained within the other.

On the right we show the intersection of these cones with the plane at $t = 0.0$. This gives us the an interpretation of the model in terms of a type of disk model discussed in section 4.7.2. The point at the centre of the circle is the projection of the apex onto this plan. Again the smaller blue circle is completely within each of the two larger circles but nether of these larger red and green circles are completely within the other. Thus two-dimensional intersection of the three-dimensional cones with a plane at constant t gives exactly the same DAG. If we view this Minkowski space-time model in terms of a disk model, each node u are represented by the points at the centre of each circle (x_u, y_u) . We then use set inclusion of section 4.2 on these circles to define the edges using circles of radius t_u centred at (x_u, y_u) .

One application where this physical viewpoint is useful is in causal set theory, one approach to quantum gravity, in which also the transitively complete DAGs created in this case provide useful insights Dowker (2006), Ilie et al. (2006), Rideout and Wallden (2009), Surya (2019).

We can, however, also work with a more general family of models producing DAGs which need not be transitively complete by combining the partial order causality constraint (4.6.3) with a cutoff in the pseudo-distance function $d^{(\tau)}$ (4.6.2). So for any chosen set of nodes $\mathcal{V} \subseteq \mathbb{M}^D$ we would define the edges as

$$(u, v) \in \mathcal{E} \quad \text{iff} \quad u_0 < v_0, \quad (v_0 - u_0) \geq \sqrt{\sum_{i=1}^{D-1} (v_i - u_i)^2} \quad \text{and} \quad d^{(\tau)}(v, u) < R. \quad (4.6.4)$$

This has some characteristic properties of what we called a DAG metric $d^{(\text{DAG})}$ in (3.3.5), that

is

$$d^{(\tau)}(x, y) \geq 0 \text{ if } x \succeq y \quad \text{non-negativity}, \quad (4.6.5a)$$

$$d^{(\tau)}(x, x) = 0, \quad \text{reflexivity}, \quad (4.6.5b)$$

$$d^{(\tau)}(x, z) \geq d_{\text{Cu}}(x, y) + d_{\text{Cu}}(y, z) \text{ if } x \succeq y \succeq z \quad \text{reverse triangle identity}. \quad (4.6.5c)$$

We could also define a closely related symmetric DAG metric where we replace $(x_i - y_i)$ in (4.6.4) by the modulus of this expression, $|x_i - y_i|$ and define this symmetry $d^{(\tau)}$ whenever we have time-like separated coordinates, that is when $(x_0 - y_0)^2 \geq r^2$. Then we would add the symmetry condition that $d(x, y) = d(y, x)$ for this symmetric version of the DAG metric. The only property which is not immediately obvious is the reverse triangle identity. **(T)**¹⁵.

The limit $R \rightarrow \infty$ will always give the transitively complete DAGs used in causal set theory. As we take $R \rightarrow 0$ we will get fewer and fewer edges until at some point we encounter a percolation transition at which point the model no longer contains a percolation cluster, that is there is no infinite DAG formed in the limit of infinite volume.

It is useful to note that in these models the time coordinate v_0 of each point gives a natural total order for our DAG. There is no need to topologically order the DAG.

4.7 Pyramid and Disk Space DAGs

(T)Mark as optional or put into appendix? **(end T)**

We can take inspiration from the Minkowski space-time and use the geometric viewpoint to create a whole family of models. Let us assume we have a set of nodes with D coordinates where one of these t we will treat as a time coordinate giving us a total order. The remaining coordinates form a $D - 1$ dimensional a spatial vector, say \mathbf{r} with spatial coordinates r_i for $i = 1, 2, \dots, D - 1$ so our space-time is $\mathbb{R} \times \mathbb{S}$ where \mathbb{S} is our $D - 1$ -dimensional space. The forwards and backwards light cones in our Minkowski space-time models were cones, the light cones of special relativity, if we looked at how they intersected a plane at constant t they gave us a circle if $D = 3$ and a $D - 1$ dimensional sphere in general. However, there is no reason why we cannot choose other shapes for these kernels. **(T)**¹⁶

Here we will consider pyramids of different sizes but all with the same shaped base. That is suppose we define a shape \mathcal{S} as a single connected volume in $D - 1$ dimensions. This will contain the origin which we will call the centre of the shape. We can translate this shape to be centred on any other point, \mathbf{x} in the $D - 1$ dimensional space. We can also scale the size of the shape by parameter R . Put another way we can define a shape of size R which must not be negative and centred on \mathbf{r} as the set of points $\mathcal{S}(\mathbf{r}, R)$ where

$$\mathbf{x} \in \mathcal{S}(\mathbf{r}, R) \text{ iff } \frac{1}{R}(\mathbf{x} - \mathbf{r}) \in \mathcal{S} \quad \text{where } R > 0. \quad (4.7.1)$$

In particular $\mathcal{S}(\mathbf{r} = 0, R = 1) = \mathcal{S}$ our defining shape. For instance, the shape for the connection kernel could be the unit square (hyper-cube in $D - 1$ dimensions) shown in Fig. 4.11 which can be described formally as

$$\mathcal{S}^{(\text{square})} = \{ \mathbf{x} \mid |x_i| < 1 \forall i \in \{1, 2, \dots, D - 1\} \}. \quad (4.7.2)$$

The forward light cones of the Minkowski space model scaled with time such that the spatial length scale was simply proportional to the time and we will do the same here. We define our

¹⁵**(T)PROOF**

¹⁶**(T)Concave?**

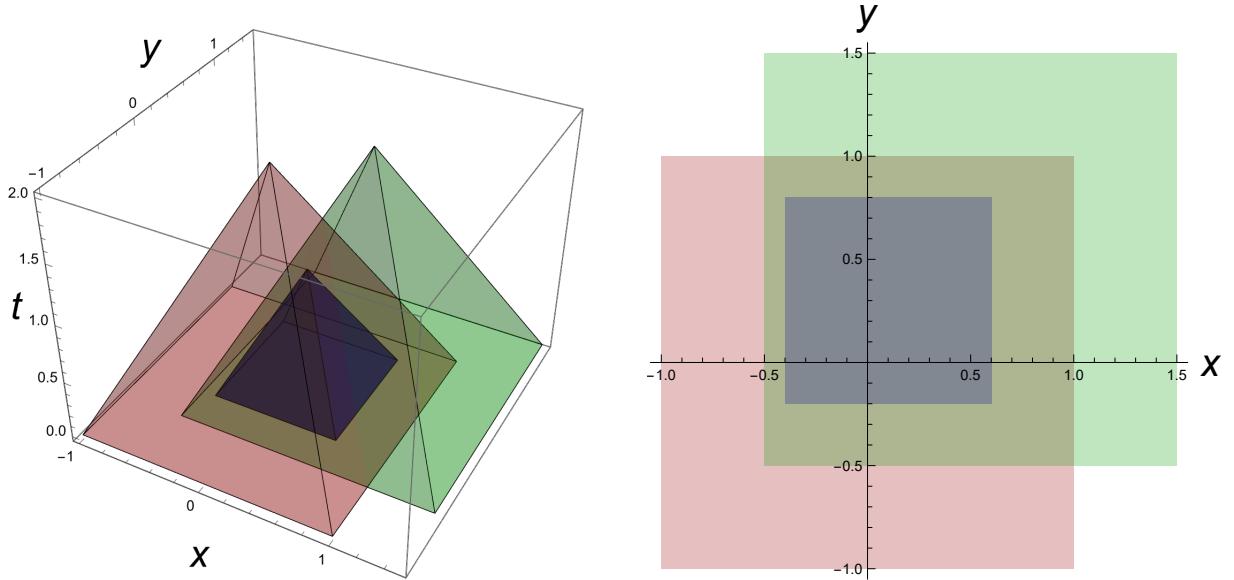


Figure 4.11: This illustrates how connection kernels of various shapes can lead to DAGs. Here we have a model where the connection kernels are pyramids with a square base in a spatial plane (here labelled x and y) with this axis of symmetry is aligned along the time axis (t). Here we show the backwards connection kernels for the three nodes whose coordinates are at the apex of each pyramid are the coordinates of three nodes in the DAG at $(t, x, y) = (2.0, 0, 0, 0, 0)$ (red), $(2.0, 0.5, 0.5)$ (green) and $(1.0, 0.1, 0.3)$ (blue). The edges can be defined using set inclusion principle (??) with an edge from one node to another if the backwards connection kernel of the source node is completely contained within that of the target node. This gives us exactly the same DAG as using the condition (4.7.5) Equally we can also take any plane at constant t less than the time coordinate of any of the points. The intersection of the backwards connection kernels gives squares on this plane and the nodes have the coordinates of the points at the centre of these squares (the (x, y) coordinates of the node at (t, x, y)). Again there is an edge from one node to another if that node's square is completely contained within the square of another node.

forward connection kernels to be “pyramids” $\mathcal{C}^+(\mathbf{x})$ with the time axis passing through the vertex and the cross-section at each time, a slice through the spatial part of space \mathbb{S} , is the same shape but which are the points $\mathbf{x} = (t, \mathbf{r}) \in \mathcal{X} = \mathbb{R} \times \mathbb{S}$ in D dimensional space-time that lie with in our shape of size t centred on \mathbf{r} . That is

$$\mathcal{C}^+(\mathbf{x}) = \{\mathbf{x}' = (t', \mathbf{r}') | \mathbf{x}' \in \mathcal{X} = \mathbb{R} \times \mathbb{S}, \mathbf{r}' \in \mathcal{S}(\mathbf{r}, t' - t)\}. \quad (4.7.3)$$

at each time t the points in the pyramid. For instance we can have a square based pyramid where the the apex is at some point (t, \mathbf{r}) which are the coordinates of the associated node in the network. Then any point (t', \mathbf{r}') is in the pyramid, the forward connection kernel, if $|r_i - x_i| < (t' - t)$ so it only contains points in the future where $t' \geq t$. The backwards connection kernel is the same shape but with time inverted relative to the apex, so

$$\mathcal{C}^-(\mathbf{x}) = \{\mathbf{x}' = (t', \mathbf{r}') | \mathbf{x}' \in \mathcal{X} = \mathbb{R} \times \mathbb{S}, \mathbf{r}' \in \mathcal{S}(\mathbf{r}, t - t')\}. \quad (4.7.4)$$

We can write the rule for connection in this square pyramid model in a language similar to the proper-time condition of Minkowski space-time. Then there is an edge (u, v) from node u to node v with coordinates $(u_0, u_1, \dots, u_{D-1})$ and $(v_0, v_1, \dots, v_{D-1})$ if

$$(v_0 - u_0) \geq |v_i - u_i| \forall i \in \{1, 2, \dots, D-1\} \text{ and } \mathbf{u} \neq \mathbf{v}. \quad (4.7.5)$$

Here we call the first coordinate u_0 the time coordinate of node u and the spatial coordinates are the remaining $(D - 1)$ coordinates¹⁷.

From these shapes we can build a pyramid in the D dimensions by rescaling the size of the shape at each time. That is we construct the forwards connection kernel from the point (t, \mathbf{x}) in our D -dimensional space scaling

4.7.1 XTent models

The is an interesting example of a pyramid/cone model that has been used in a slightly different way to give a DAG with fewer edges than we would define using the rules we have set out so far. The XTENT MODEL (Renfrew and Level, 1979) was defined to enable archaeologists to get a feeling for how different settlements or regions would be dominated/influenced by settlements in the archaeological record. In the XTent model, every site v has a location and size so site v is represented by the pair $(s_v, \mathbf{v}) \in \mathbb{R}^+ \times \mathbb{R}^2$ where \mathbf{v} is the geographical location of the site and s_v is the size of the site. Assuming simple Euclidean measures of distance, we draw a cone down to the ‘ground’, the surface $s = 0$, of height s_v with axis passing through the apex at (s_v, \mathbf{v}) and the point $(0, \mathbf{v})$. These cones are the ‘tents’ which give the model its name. The angle θ of the cone is the same for all sites so this is a parameter of the model. This means each cone gives a disk of radius $r_v = s_v \tan(\theta)$ on the ground. The idea is that these disks on the ground give a region of influence of each site and the θ parameter gives a measure of the strength of influence being discussed. That is, small θ gives regions where sites have a very strong influence and small θ indicates regions where the influence is much weaker. At this point we have a Minkowski space-time model of a DAG where the time coordinate is given by site sizes, a model in “space-size” geometry. Equivalently this can be viewed as a set inclusion DAG model defined by the cones in three-dimensional space-time/size space or the disks on the two-dimensional ground. That is the INFLUENCE DAG $\mathcal{D}(\theta)$ defined by the XTent model for a given value of θ is the set of sites as the set of nodes \mathcal{V} and the link set is formally

$$\mathcal{E}(\theta) = \{(v, u) | s_u \tan(\theta) > d(v, u), s_v < s_u\}, \quad (4.7.6)$$

where $d(v, u)$ is the distance from site u to site v . We follow the space-time convention and always draw edges from the apex of the cone to sites in the forward connection kernel defined by these cones. That is edges must be from larger sites to smaller sites but that condition is not sufficient. In fact this is a transitively complete DAG and so it also defines a poset on the settlements. As θ increases we add more an more edges until as $\theta \rightarrow \pi/2$, so $\tan(\theta) \rightarrow \infty$ we get a complete DAG where there is an edge between every pair of sites with the edge direction always from larger to smaller sites. Interestingly, this DAG of site-site influence was not the finishing point of the original paper. For each site v , the DAG of influence for some value θ was used to define a single dominating site $D(v; \theta) \in \mathcal{V}$ that acted as the site which dominated site v . What we are doing in terms of the influence DAG is that we are removing all but one incoming edge to each site. The one edge we retain is the dominating site. The way the model works is that the site which dominates v is not always the site u with the largest site that has a connection (v, u) in the influence DAG. More formally, $D(v; \theta)$ is the node that for a given θ has the highest cone height at the location of site v . That is $D(v; \theta)$ is the site u that maximises $s_u \cdot [1 - (d(u, v)/r_u)]$ provided this is positive where $r_u = s_u \tan(\theta)$ is the radius of the disk of influence of node u . Should $\max_u(s_u \cdot [1 - (d(u, v)/r_u)])$ be negative then $D(v; \theta) = v$ i.e. no other site dominates v .

¹⁷In fact there is an amusing way we can generalise the concept of proper time for the case of these models where the connection kernels are square pyramids (but not for more general shapes). We may define a generalised proper time $\tau_p = [|u_0 - u_0|^p - \sum_{i=1,2,\dots,D-1} |v_i - u_i|^p]^{1/p}$ which is a space-time generalisation of the well-known L_p or Minkowski metric. Again we have an edge whenever τ_p is real with the edge running from node with the smaller time coordinate to node with the larger time coordinate.

The model can be improved in several ways, such as taking into account terrain using GIS software to calculate the distance between points (Bevan, 2010). Of course as a model of social, political or economic influence, this model is incredibly crude and likely to be unrealistic in so many ways. However, the original context of Renfrew and Level (1979) was neolithic structures on Malta. As a physicist I imagine that the organisation was much ‘simpler’ in earlier times so a simpler model such as the XTent model might be a better fit for prehistoric eras than modern times. Alternatively, there is incredibly little information to go on in the archaeological record compared with data on society from the modern era so perhaps at this level of information the use of a sophisticated model is unjustified and will be misleading. One could run this around and state that despite the vast amount of information we have in a modern context, this wealth can be misleading, covering, large holes or biases in the data. We can never collect all the information we need for an exact model. So, ancient or modern, this type of modelling does not aim for perfection. Rather, it aims to capture key aspects of the problem, to highlight the role each feature plays. The model should be part of the conversation, never the end of the conversation. It lays out the assumptions, good or bad, clearly, allowing others to criticise or improve. So there is a lot to be learnt from those modelling archaeological and historical data, for example see Brughmans and Peeples (2023) or Graham et al. (2015) for modelling including the use of general networks in such circumstances.

4.7.2 Pyramids from Disk Spaces

We defined disk spaces and their associated DAGs in section 4.2.1. This was done using $(D-1)$ -dimensional disks $\Delta(\mathbf{x}, r)$ defined to be closed hypervolumes of some $(D-1)$ -dimensional space which has a spatial size characterised by $r \in Rbb$. We can see that these disks can be the projections of pyramids P_Δ in a D -dimensional space time $\mathbb{R} \times \mathbb{X}$ onto a surface of constant time $t = 0$. To do this our pyramids are defined to give a disk $\Delta(\mathbf{x}, r - t)$ on a hypersurface of constant time t . That is the apex of each pyramid is at the point $(t = r, \mathbf{x}) \in \mathbb{R} \times \mathbb{X}$ in our space-time. More formally

$$P_\Delta(\mathbf{x}, r) = \{(h, \mathbf{y}) | \mathbf{y} \in \Delta(\mathbf{x}, r - h)\} \in \mathbb{R} \times \mathbb{X}. \quad (4.7.7)$$

So, just as with the XTent model, we can either work with D -dimensional pyramids in space-time or their $D - 1$ projections onto a spatial surface, the disks of a disk space. In both cases we can see that one way to view the order and the DAGs inherent in these models is to take a geometric view and use set inclusion to define the edges in a DAG with the nodes representing these D -dimensional shapes in space-time (e.g. they are linked to the coordinates of the apex of the pyramids forming the forwards and backwards connection kernels). In the literature these are usually defined in terms of algebraic expressions but one can move between the two viewpoints and use whichever provides the simplest approach to any given problem. The Minkowski space and cube space models are examples which are naturally defined in terms of D -dimensional space-time and are usually defined in terms of their algebraic properties, hypercones and hypercubes being the pyramids respectively.

However, we have also seen that these models can also be interpreted in terms of a geometric picture in terms of regular closed shapes in a $D - 1$ -dimensional space, with each node associated with the centre one shape, hyperspheres circles for Minkowski space DAGs and $(D - 1)$ -dimensional hypercubes for cube space. Again set inclusion gives a natural way to define the DAG in this lower dimensional spatial viewpoint. There are several models which are defined in the literature in this spatial view point.

Disk Spaces and Box Spaces

(T)EDIT DOWN (end T)

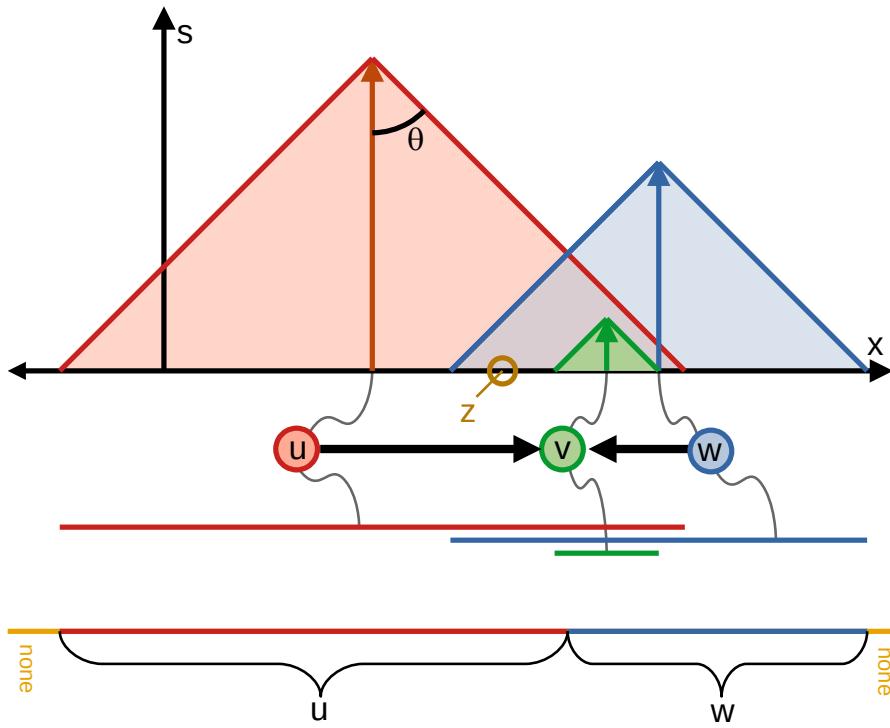


Figure 4.12: An example of the XTent model. In this example we have one spatial dimension, represented by the horizontal axis x , the vertical axis is the ‘size’ s and we use Euclidean geometry for simplicity. Each site u , v and w is represented by a cone, the ‘tent’ of this model, centred at the location of the site and with height equal to the size of the site. These cones are shown in the top part of the figure. The model has one parameter θ which is the angle subtended by the axis of symmetry of each cone and the sides of the cone. Here we choose $\theta = 45^\circ$ for simplicity. If the apex of the cone from a site, for example site v , falls within the cone of another site, for example u , then we assign an edge (v, u) from larger site u to smaller site v . This represents the fact that the smaller site is under the influence of the larger site, the smaller site is part of its political or social domain the influence of the larger site. At this point this defines a set inclusion DAG, and so this is transitively complete, in which the nodes are sites and edges point from sites subjugated sites to those larger site which have a controlling influence over the smaller site. This DAG is shown just below the x axis. Below the DAG is the projection of the cones onto the x axis or ‘ground’ in this model. We can also interpret the model as set inclusion applied to these disks (here lines, one-dimensional disks).

In the original model, there was a further criteria. That is, if a small site was dominated by more than one site, then it would be placed under the influence of the site whose cone had the largest height at location of the smaller site. For example here v is within the cone of both v and w but the cone of w is higher at the location of site v so we have that $D(v) = w$.

The model allows us to assign every point in space to be under the influence of one particular site or none. We treat the size of generic points in space to be zero and use the same criteria. So the point z is under the cones of both v and w but here the cone of w is larger so z is assumed to be in the zone of control of site w . Points to the far left and far right are not under any cone and so are not in the zone of control of any site shown here. These zones of influence are shown in the line at the bottom.

It is clear, as noted in Suzuki et al. (2019) in section 3.1, that Co_D of Bollobás and Brightwell (1991), i.e. D -dimensional Minkowski space-time is an example of a disk space. Look at a space-like hypersurface, say at constant time later than the time of all points (events). WLOG choose this surface to be at $t = 0$, then the light cones of an event at (t, \mathbf{x}) (t is negative in this definition) intersect the time-like surface to define $(d - 1)$ -dimensional spheres of radius $r = -ct$ (choosing units where speed of light $c = 1$) centred at \mathbf{x} and these are the disks or generalised balls in this model. The hypersurface is a Euclidean $(D - 1)$ -dimensional space

and thus set inclusion of these light cones and the Euclidean distance between radii give the same poset. However, note that we have to impose a distance and metric on the cone space or at least its spatial hypersurface which, while natural, is not needed. In some sense the set inclusion definition is all we need.

The cube space Cu_D of Bollobás and Brightwell (1991) needs a different perspective. Now it appears that we project the forward connection kernels, hypercubes in this model onto a (flat?) plane which cuts all the axes at positive values. That is we project onto a plane perpendicular to any vector \mathbf{p} where $p_i > 0$ for all $i = 1, 2, \dots, D$. This defines triangles in $D = 3$ and in general simplices with D vertices on the $(D - 1)$ dimensional hypersurface. This is because each line from a point which is parallel to an axis defines one edge of the forward connection kernel and each of these edges gives a single point on the $(D - 1)$ dimensional hypersurface. This is the content of theorem 1 in Suzuki et al. (2019) (see appendix for detailed proof and Fig 4a for $D = 3$ example) where cube space is called ORDER EMBEDDING, (Vendrov et al., 2016, Suzuki et al., 2019).

In general we never needed to limit ourselves to cones and hypercubes for the shape of the forward connection kernel (light cone) in these models. We could easily define the forward connection kernel using a constant set of $n \geq D$ or more vectors $\mathbf{w}^{(i)}$, and these would be the edges of the forward connection kernels from any point, i.e. $\mathbf{x} + \lambda\mathbf{w}^{(i)}$ for $\lambda > 0$. The limit of $n \rightarrow \infty$ presumably interpolates between cube $n = D$ and cone space $n = \infty$.

Some comments. There is no distance/metric in the cube space so we cannot use the distance measure on the disk space project, only set inclusion. We could change the angle subtended by the boundaries, i.e. change $c = \tan(\theta)$ in the $r = -ct$ relationship. This is exactly what happens in the XTENT MODEL (Renfrew and Level, 1979). The poset changes as we can see that in the example in Fig. C.6 as we reduce c from $c = 1$ the first edge to go will be the (A, C) edge.

4.8 The Price model

The PRICE MODEL was put forward by Derek de Solla Price in 1976 (Price, 1976) as a model of citation networks. Price was inspired by earlier ideas on fat-tailed distributions in general, such as Merton’s discussion of the Matthew effect in science (Merton, 1968, 1988), as well as his own observations such as the physical height of the editions of a journal when placed in stacks by year. In particular, Price wanted a model that could explain why the data showed that the number of papers $n(k^{(in)})$ with a given number of citations $k^{(in)}$ seems to follow a power-law distribution Price (1965a,b, 1976) for large degree¹⁸ as illustrated in Fig. 4.14.

To explain these fat-tails Price (Price, 1976) proposed a model in which a newly published paper cites older papers with a probability proportional to the number of existing citations each older paper has at that time. Price called this the principle of CUMULATIVE ADVANTAGE — the more citations you have, the more likely you are to be cited in the future. Price was inspired (Price, 1965a) by Merton’s application of the Matthew effect (Merton, 1968, 1988) to the bibliometric context. This ‘rich get richer’ principle was new for the network context of Price but had been used in many other contexts and appears under different names:

- The Pareto principle or the 80–20 rule, 19th cent. .
- The Matthew effect, after the quote from Matthew’s gospel “For everyone who has will be given more”.

¹⁸It is unclear if other distributions such as a log-normal distribution, give a “better” fit to this distribution in terms of some statistic. The key observation is that citation network invariably show these types of fat-tailed distribution.

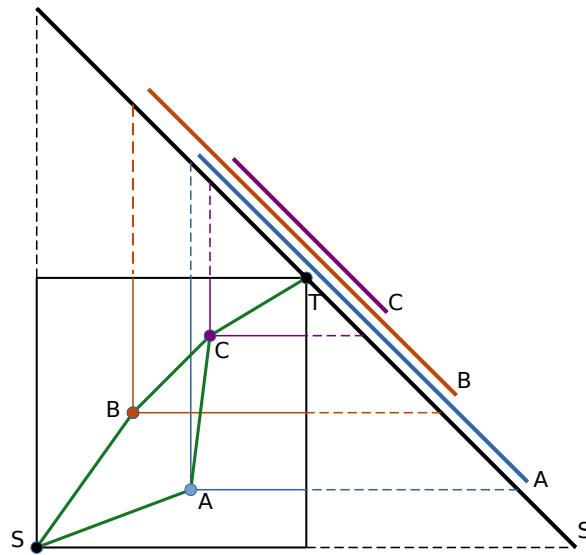


Figure 4.13: An illustration of how a cube space in two dimensions is related to a disk space where disks are in one dimension. The interval runs from the global sink S at $\mathbf{0} = (0, 0)$ to the global sink, the target node T at $\mathbf{1} = (1, 1)$. The interval is the square shown in solid black lines. There are three points inside the interval shown as circles with labels A , B and C . The green line connecting points show the relations after transitive reduction (the Hasse diagram of the poset). The solid lines leaving each point show the boundaries of the forward connection kernel.

The disk space is shown as the projection of the forward connection kernels onto the plane perpendicular to the $(1, 1)$ vector which passes through the target (global sink) node T at $(1, 1)$. The continuation of the boundaries of the forward connection kernels are shown as dashed lines of the same colour as the associated point. Each point corresponds to a disk shown as a one-dimensional line of length equal to its radius. These are the solid lines running diagonally from top left $(0, 2)$ to bottom right $(2, 0)$. The solid black diagonal line shows the disk for the source node. The other disks are displaced slightly in the direction of the $(1, 1)$ vector in order to show them clearly. These are the solid lines with labels and colours matching the corresponding point. The disk for the global sink node T is just a single point and the same circle is used to indicate this

- The Yule-Simon distribution (Yule, 1925) (1925) emerging from a description of biological taxa and subtaxa.
- Simon's model for the frequency of words in text (Simon, 1955).

The model reappeared in 1999 as the BARABÁSI-ALBERT MODEL which creates undirected networks with a power-law degree distribution. Barabási and Albert compared their undirected version of the model to a much wider range of networks drawn from many different contexts other than bibliometrics. It is their term, PREFERENTIAL ATTACHMENT, which is now used to what Price called cumulative advantage.

A rough outline of the Price model is as follows.

1. Start at ‘time’ $t = t_{\text{init}}$ with an initial directed acyclic graph, $\mathcal{D}(t_{\text{init}}) = (\mathcal{V}(t_{\text{init}}), \mathcal{E}(t_{\text{init}}))$ containing $t_{\text{init}} = |\mathcal{V}(t_{\text{init}})|$ nodes¹⁹.
2. Increment time by one, $t \rightarrow t + 1$.
3. Add one new node v with a label $t(v)$ given by the time this node was added to the network, so $t(v) = t$.

¹⁹For simplicity, we will assign distinct integer labels $t(u)$ to nodes $u \in \mathcal{V}(t)$ running from 1 to t . We make sure that edges always run from a high-value label to a low-value label.

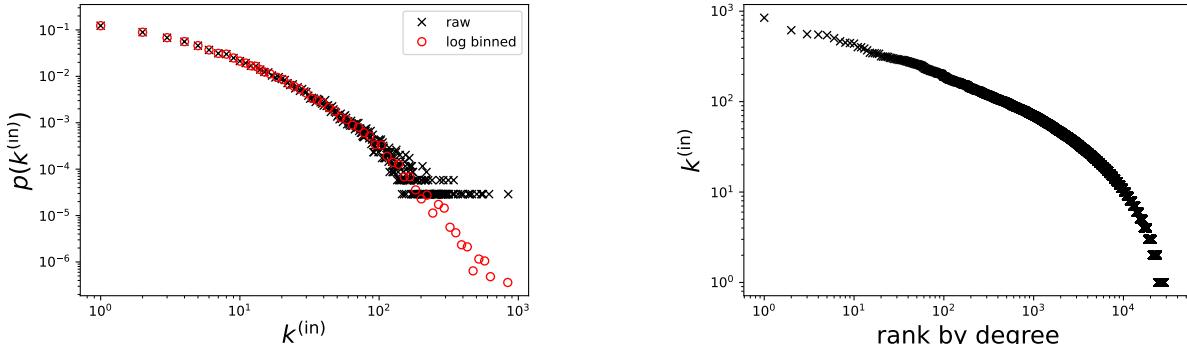


Figure 4.14: Illustration of the fat-tail degree distribution in the citation network of the first ten years of data from the hep-ph arXiv preprint repository. This network has 34,546 nodes and 420,921 edges so $\langle k^{(\text{in})} \rangle = 6.1$. On the left is the in-degree distribution on a log-log plot, both raw and binned. On the right we plot the distribution as the number of edges for a given node against the rank of the node in terms of in-degree, a ZIPF PLOT. Data from the KDD Cup 2003 (Gehrke et al., 2003, KDD cup).

4. Choose m nodes from the existing network $\mathcal{D}(t-1)$ of $(t-1)$ nodes with probability $\Pi(k, t)$ which is linear in-degree $k^{(\text{in})}$ of each node. That is we choose $\Pi(k, t) \propto ak^{(\text{in})} + b$ for constants a and b . We then connect one edge from each of the existing nodes to the new node t .
5. Repeat from (2).

The normalisation of the edge attachment probability $\Pi(k, t)$ gives a relation between a and b (as we will note below) so this model has two parameters²⁰. These rules are illustrated in Fig. 4.15. In practice there are various details that need to be dealt with which we will address below in section 4.8.1.

In the original citation network context of Price, each node is a document. The label $t(u)$ given to each node is the publication date of that document but given in ‘ranked time’. That is if $t(u) = 1$ then node u represents the oldest document, if $t(u) = 2$ then node u represents the second oldest document, and the most recent document added to the network is equal to the number of nodes $N(t)$ in the network at that time, so $t(u) = N(t)$. This ranked-time label also gives a natural total order for the nodes in this DAG.

Of course this model is a gross simplification of the process of citation. Documents are published at a variety of times, often coming in bunches, not in a nice even order as given by the ranked ‘time’ parameter t used here. In reality the length of the bibliography of each document²¹ is not a fixed number m . Nevertheless, the Price model shows that a simple process is sufficient to create a fat-tailed distribution of the type shown in Fig. 4.14.

It is worth emphasising that while Price’s original version of this network and our own discussions have used the example of citation networks to motivate the model, it could be used for any context where a DAG was needed if that was felt to be appropriate.

The final comment is that the Price model is stochastic but it is *not* the same as a randomised version with the same degree distribution. For instance, we could apply the DAG configuration model of section 6.3 and a network with different properties. While the growth of the network in the Price model over time is stochastic, it is performed in a very particular way. For instance

²⁰In the first versions of these models, one of these parameters was often fixed, e.g. Price (1976) worked with $b = 1$ while Barabási and Albert (1999) set $b = 0$.

²¹The number of papers in a bibliography, the out-degree, is often a fat-tailed distribution in its own right.

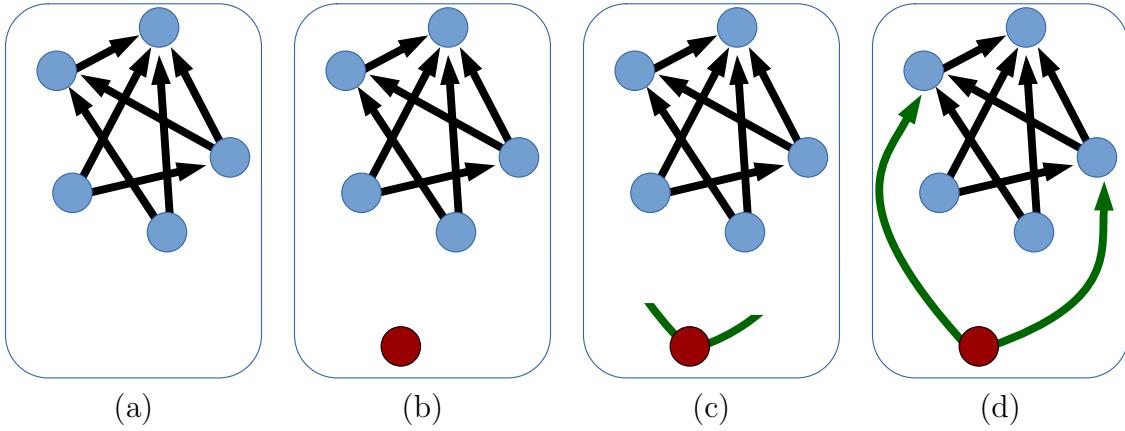


Figure 4.15: An illustration of how the network grows in the Price model. The initial directed acyclic network, $\mathcal{D}(t = 5)$, is shown in (a) with blue circles for the nodes. First in (b) a new node, shown in red, is added; here it is the sixth oldest node. Next in (c) m new edges are attached to the new node, illustrated here for $m = 2$. We then we have to choose which nodes in the existing graph, the blue nodes in $\mathcal{G}(t = 5)$, are attached to the new red node at the other end of these m new edges. In the Price model this is done stochastically using cumulative advantage (preferential attachment) and we show one possible outcome. We arrive at the final network shown in (d), which is the network $\mathcal{G}(t = 6)$. The cycle is then repeated.

in the Price model the nodes with high in-degree (edges from newest to oldest) are the earliest nodes in the network since they had more opportunities to gather extra edges (see (4.8.35)). Most network measurements will show a difference between this randomised network and the original Price/BA network.

4.8.1 Issues with the definition of the Price model

When making an actual realisation of the Price model, the loose rules outlined above are not precise enough and various decisions need to be made.

First when choosing nodes as the target for one of the m new edges, the same target could be chosen more than once for the same new source node. If we allow this we would have a network with multiedges which we could interpret in many ways e.g. just treat a multiedge as a single edge. Alternatively we need to add an additional test, in code or analytic equations, to stop this happening. In practice one can see, either analytically or numerically, that the chance of multiedges happening is small and tends towards zero as the number of nodes rise, relative to the frequency of choosing m different targets for each new source node. Basically we would chose a given node with degree k once out of m times with probability roughly $m\Pi(k, t)$ and twice with probability roughly $(m(m - 1)/2)(\Pi(k, t))^2$. Even for the node with the largest degree in the network, we find the probability $\Pi(k, t)$ dies off as a power of t (see below) and so this possibility of multiedges dies away as the network grows. So in most contexts, it makes little difference if we allow multiedges or not.

Another issue is that the initial graph needs to be specified. At first sight, this is unlikely to play a large role once the network is much bigger than the initial graph. However, as mentioned above, the highest degree nodes at any time tend to be the oldest (see (4.8.35)). So if we are interested in the properties of the largest nodes, these tend to be nodes which were present in the initial graph. Therefore, the precise choice of initial graph does alter measurements of the largest nodes at any point in time. For most other measurements, the choice of the initial graph has no noticeable effect so we can make any choice that is convenient.

One response to these uncertainties is to recast the model in a more precise way, and this

is an approach found in the mathematics literature where rigorous results, usually for infinite networks, can then be obtained. For instance the CHUNG-LU RANDOM GRAPH (Chung and Lu, 2001, 2002a,b, 2006, Fasino et al., 2020) can be cast as a random DAG model(\mathbf{T})²² and is designed to look at random networks with a given degree sequence, typically those with a power-law degree distribution. Another version of the Price model is the RANDOM CIRCUIT MODEL (Arya et al., 1999) (but also see (Broutin and Fawzi, 2012, Janson, 2023) and references therein) which was always designed as a DAG and is really about logical flow, digital circuits with gates rather than wires and resistors.

4.8.2 Price Model Mean Field Master Equation

MASTER EQUATIONS describe the evolution a system by describing how the probability for being in any one state at one time evolves given the probability of moving from one state to another. Here we will build the master equation for the degree distribution in the Price model.

To reach something that is analytically tractable, we have to make further simplifying assumptions. The model is stochastic, that is each time we run the model a slightly different set of edges are added. For instance an edge will rarely be added to a low degree node but it does happen sometimes. The outcome of one update at time t could be any one of an extremely large number of different networks, a set $\{\mathcal{D}(t+1)\}$. The probability of reaching any one of these networks is easily found in principle from the rules of the model but the expressions are far too complicated and there are far too many possible networks for these results to be useful. So we do not ask questions about the exact network we arrive at, the $\mathcal{D}(t+1)$, but rather we ask about typical general properties of such networks. In terms of mathematics, we are going to take an ensemble average and ask what is the average value of some quantity Q . To be clear we ought to adopt a notation such as

$$\langle Q(t) \rangle = \sum_{G \in \mathcal{G}(t)} p(G) Q(G). \quad (4.8.1)$$

Here $\mathcal{G}(t)$ is the set of possible networks at time t and $p(G)$ is the probability that we arrived at network G . The angular brackets $\langle \dots \rangle$ indicate that we are taking this ensemble average.

Finding the average value $\langle Q(t) \rangle$ is only useful if $\langle Q(t) \rangle$ is typical of the result we get when we measure this quantity $Q(t)$ in any *one* run of the model. If the model produces a wide range of different results for $Q(t)$ every time we run the model then the average $\langle Q(t) \rangle$ is not going to be a very useful indicator. Fortunately in many cases we find that individual results are usually close to the ensemble average with little variation for large systems (e.g. the central limit theorem applies). This use of average values in our equation and replacing $\langle \langle f(x) \rangle \rangle$ by $f(\langle \langle x \rangle \rangle)$ for some function f of a stochastic variable x , is known as a MEAN FIELD APPROXIMATION. So we will build an approximate equation which states the average value for the number of nodes with in-degree $k^{(\text{in})}$ at time t , depends only on the *average* value of the degree distribution at earlier time steps.

In our case we are interested in the degree distribution so we want to find $\langle n(k^{(\text{in})}, t) \rangle$, the number of nodes with in-degree $k^{(\text{in})}$ at time t averaged over all ensembles. In fact the normal practice is to drop the angular brackets $\langle \dots \rangle$ so here we will use the standard notation of $n(k^{(\text{in})}, t)$ for $\langle n(k^{(\text{in})}, t) \rangle$.

The master equation will be an expression for the number of nodes of degree k at the next time step ($t+1$), that is $n(k, t+1)$, given the distribution at previous times. Here the simple rules for the model mean that we only need to know the degree distribution at the current time

²²(T) REALLY?

t , i.e. $n(k, t)$, to predict what the results of adding m edges will be²³. Suppose that $\Pi(k, t)$ is the probability that one of the new edges at time t is connected to an existing node with degree k . The MASTER EQUATION for the average degree distribution in the Price model is then, approximately,

$$\begin{aligned} n(k^{(\text{in})}, t+1) = & \quad n(k^{(\text{in})}, t) && [\text{Start with this number}] \\ & + m\Pi(k^{(\text{in})}-1, t)n(k^{(\text{in})}-1, t) && [\text{Add edge to degree } (k^{(\text{in})}-1) \text{ node}] \\ & - m\Pi(k^{(\text{in})}, t)n(k^{(\text{in})}, t) && [\text{Add edge to degree } k^{(\text{in})} \text{ node}] \\ & + \delta_{k^{(\text{in})}, m} && [\text{The one new node has } m \text{ edges}] \end{aligned} \quad (4.8.2)$$

We say approximate because the average number of edges added to nodes of degree $(k^{(\text{in})}-1)$ at time t in the model as we have defined it is only roughly equal to the average number of $m\Pi(k^{(\text{in})}-1, t)n(k^{(\text{in})}-1, t)$ e.g. we are ignoring multiedges.

(T)²⁴ (T)²⁵

This master equation is an example of a difference equation for the discrete variables $k^{(\text{in})}$ and t . A difference equation is the discrete-variable equivalent of a differential equation in continuous variables. The tricks used to solve such difference equations mimic those used in differential equations, for instance see Bender and Orszag (1999). For example, a differential equation requires boundary conditions to be specified and the same is true here. In our case we note immediately that physically (if not mathematically) we have that no node can have negative in-degree (T)²⁶ so $n(k^{(\text{in})}, t) = 0$ if $k^{(\text{in})} < 0$. Equally, as we phrased this in terms of an evolution, it is natural to specify a boundary condition in time, that is the shape of the degree distribution at some initial time.

Before we solve this difference equation we have to deal with a problem in our definition of cumulative advantage given at the start of this section. The rough definition given suggested that new citations to papers would arrive in proportion to the current number of citations, so $\Pi(k, t) \propto k^{(\text{in})}$. However, the problem with this is that all new papers start with zero in-degree and so these would never attract a citation from any later paper and would remain without a citation, clearly not realistic. So in practice we choose

$$\Pi(k^{(\text{in})}, t) = \frac{1}{Z(t)}(k^{(\text{in})} + \mu) \quad (4.8.3)$$

for some positive constant μ which is a measure of the inherent attractiveness to later citations of all papers. In his original work, Price choose $\mu = 1$ for convenience (Price, 1976).

The $Z(t)$ is the appropriate normalisation which we now need to fix. First we note that the number of edges grows linearly as

$$E(t) = m(t - t_{\text{init}}) + E(t = t_{\text{init}}). \quad (4.8.4)$$

²³This means we have a MARKOV PROCESS, i.e. one where you do not need the whole history of the evolution of the system to predict its future evolution.

²⁴(T)One reason is that in any one run of the model, the term we need is $m\Pi(k^{(\text{in})}-1, t)n(k^{(\text{in})}-1, t)$ for each case. As we are writing the equation for the average value $n(k^{(\text{in})}, t+1)$ (remember we dropped the $\langle \dots \rangle$ from our notation) the left-hand side should really have contributions from terms such as $\langle k^{(\text{in})}n(k^{(\text{in})}-1, t) \rangle$. However this is not the same as $k^{(\text{in})}\langle n(k^{(\text{in})}-1, t) \rangle$.

²⁵(T)In part because we are ignoring possible effect of the inclusion or exclusion of multiedges which would require additional terms along the lines of $(m(m-1)/2)\Pi(k^{(\text{in})}-2, t)\Pi(k^{(\text{in})}-2, t)n(k^{(\text{in})}-2, t)$. Since these multiedge terms are an $O(m\Pi)$ correction they are negligible in the long time limit and we will leave them to one side. We look at them in more detail below in section ?? only when we have sketched out the basic solution.

²⁶(T)In fact every node added in the Price model always has at least m edges as these are added immediately. So only nodes in the initial network can ever have degree between 0 and m so one could always assume an initial network where all nodes have m or more edges. In that case $\Pi(k, t)n(k, t) = 0$ for all $k < m$.

For simplicity we will assume that²⁷ we can start our model from $N(t = t_{\text{init}}) = t_{\text{init}}$ and $E(t_{\text{init}}) = mt_{\text{init}}$. The normalisation of $\Pi(k^{(\text{in})}, t)$ then requires that

$$1 = \sum_{i \in \mathcal{V}} \Pi(k_i^{(\text{in})}, t) = \frac{1}{Z(t)} \sum_i (k_i^{(\text{in})} + \mu) = \frac{1}{Z(t)} (E(t) + \mu N(t)) \quad (4.8.5)$$

$$\Rightarrow Z(t) = (E(t) + \mu N(t)) = (m + \mu)t. \quad (4.8.6)$$

With this the master equation becomes

$$\begin{aligned} n(k^{(\text{in})}, t+1) &= n(k^{(\text{in})}, t) + m \frac{((k^{(\text{in})} - 1) + \mu)}{(m + \mu)t} n(k^{(\text{in})} - 1, t) \\ &\quad - m \frac{(k^{(\text{in})} + \mu)}{(m + \mu)t} n(k^{(\text{in})}, t) + \delta_{k^{(\text{in})}, 0} \quad \text{for } k^{(\text{in})} \geq 0. \end{aligned} \quad (4.8.7)$$

If we write the master equation in terms of the probability distribution

$$p(k^{(\text{in})}, t) = n(k^{(\text{in})}, t)/N(t) \quad (4.8.8)$$

we find that

$$\begin{aligned} (t+1)p(k^{(\text{in})}, t+1) - tp(k^{(\text{in})}, t) &= \\ &+ \frac{m}{(m + \mu)}(k^{(\text{in})} - 1 + \mu) p(k^{(\text{in})} - 1, t) \\ &- \frac{m}{(m + \mu)}(k^{(\text{in})} + \mu) p(k^{(\text{in})}, t) + \delta_{k^{(\text{in})}, 0} \quad \text{for } k^{(\text{in})} \geq 0. \end{aligned} \quad (4.8.9)$$

4.8.3 Price Model Master Equation Solution

Here we will search for a solution of the master equation (4.8.9) for large times, equivalently for large networks since the number of nodes always growing as²⁸ $N(t) = t$. The idea is that $n(k^{(\text{in})}, t)$ must evolve too but there will be a trivial overall growth factor which reflects the growth in the number of nodes $\sum_k n(k^{(\text{in})}, t) = N(t) = t$. So we factor this out by working with the probability distribution $p(k^{(\text{in})}, t) = n(k^{(\text{in})}, t)/N(t)$ and we search for a stable form for the probability distribution

$$\lim_{t \rightarrow \infty} p(k^{(\text{in})}, t) = p_\infty(k^{(\text{in})}). \quad (4.8.10)$$

Assuming such a form exists in the long time, large N limit, we then have that

$$\begin{aligned} p_\infty(k^{(\text{in})}) &= +\frac{m}{(m + \mu)}(k^{(\text{in})} - 1 + \mu) p_\infty(k^{(\text{in})} - 1) - \frac{m}{(m + \mu)}(k^{(\text{in})} + \mu) p_\infty(k^{(\text{in})}) \\ &\quad + \delta_{k^{(\text{in})}, 0} \quad \text{for } k^{(\text{in})} \geq 0. \end{aligned} \quad (4.8.11)$$

The factor of N in (4.8.9) has cancelled and the form of (4.8.11) does indeed suggest that there is a time and N independent $p_\infty(k^{(\text{in})})$ solution as we assumed.

It is possible to find an exact solution for the difference equation (4.8.11) but this is unusual, typically such master equations are too complicated for exact solutions. We will first use an approach which can be used in many situations where there is no exact solution for the difference equation and which is sufficient to reveal tail of the degree distribution in this model. So let us consider $k^{(\text{in})}$ large in (4.8.11). That means we can ignore the low $k^{(\text{in})}$ boundary

²⁷This assumption can be relaxed but this is left as an exercise for the reader. With a suitable definition of time coordinates and a suitable initial condition, this may also be implemented numerically.

²⁸We may always choose to count time steps t such that this is true from some initial time t_0 equal to the number of nodes in the initial network G_0 .

term $\delta_{k^{(\text{in})},0}$. We also see that the remaining terms on the right hand side of (4.8.11) form an approximation of a derivative in $k^{(\text{in})}$ as follows. First in terms of a deviation $\Delta = 1$ we have

$$p(k^{(\text{in})}) = \frac{m}{(m+\mu)} \frac{(k^{(\text{in})} - \Delta + \mu)p(k^{(\text{in})} - \Delta) - (k^{(\text{in})} + \mu)p(k^{(\text{in})})}{\Delta} \quad \text{for } k^{(\text{in})} > 0 \quad (4.8.12)$$

If we now argue that for large $k^{(\text{in})}$ the Δ interval can be considered to be small so we approximate the master equation for large $k^{(\text{in})}$ using $\Delta \rightarrow 0$ to find that

$$\Rightarrow p(k^{(\text{in})}) \approx -\frac{m}{(m+\mu)} \frac{\partial((k^{(\text{in})} + \mu)p_\infty(k^{(\text{in})}))}{\partial k^{(\text{in})}} \quad k^{(\text{in})} \gg 1. \quad (4.8.13)$$

A PDE such as this is much more familiar than the discrete difference equation form of the master equation and we can solve this in a variety of ways. As we are looking for a power law, the simplest way is to make the ansatz that $p(k^{(\text{in})}) = A(k^{(\text{in})})^{-\gamma}$ and to substitute this into (4.8.13) to see if we can make this form satisfy the equation. If we do this we find a solution to (4.8.13) of the form

$$\lim_{t \rightarrow \infty} p(k^{(\text{in})}, t) = p_\infty(k^{(\text{in})}) \propto \frac{1}{(k^{(\text{in})})^\gamma}, \quad \gamma = 2 + \frac{\mu}{m} \quad (4.8.14)$$

From this we see that the tail of the degree distribution is a power law with power $\gamma > 2$.

We can check our solution numerically. In this case we simulate the ensemble average by making a number of runs for the same parameter values, stopping at the same time but each time changing the random numbers used to make the probabilistic decisions. The ensemble averages are approximated by making averages over the networks created at the end of the runs. Of course we can never access all possible networks but for many quantities the numerical simulations will work quickly to enough to produce examples to provide a good approximation, though this ought to be checked numerically. In the same way we can never run the simulation for infinite time but the hope is a ‘long’ time will be practical and again checking what time is long enough is a trickier part of the numerical simulation of this and many other stochastic models. An example of is shown in Fig. 4.16.

It is also possible to solve the master equation (4.8.11) exactly as we will now see. First focus on the case $k^{(\text{in})} > 0$ and rearrange the equation to place the $p(k^{(\text{in})})$ factors on the left and $p(k^{(\text{in})} - 1)$ on the right as follows

$$\left(1 + \frac{m}{(m+\mu)}(k^{(\text{in})} + \mu)\right) p(k^{(\text{in})}) = \frac{m}{(m+\mu)}(k^{(\text{in})} - 1 + \mu)p(k^{(\text{in})} - 1) \quad \text{for } k^{(\text{in})} > 0 \quad (4.8.15)$$

so that

$$\frac{p(k^{(\text{in})})}{p(k^{(\text{in})} - 1)} = \frac{(k^{(\text{in})} - 1 + \mu)}{(k^{(\text{in})} + \mu + 1 + (\mu/m))} \quad \text{for } k^{(\text{in})} > 0. \quad (4.8.16)$$

These type of difference equations have a natural solution in terms of GAMMA FUNCTIONS $\Gamma(z)$ which is a generalisation of the factorial notation for integers, see section A.3, The Gamma function can be defined from $\Gamma(z+1) = z\Gamma(z)$ and $\Gamma(z=1) = 1$. From this we can deduce that if a function $f(z)$ satisfies

$$\frac{f(z)}{f(z-1)} = \frac{z+a}{z+b} \quad (4.8.17)$$

then the solution for f is

$$f(z) = A \frac{\Gamma(z+1+a)}{\Gamma(z+1+b)} \quad (4.8.18)$$

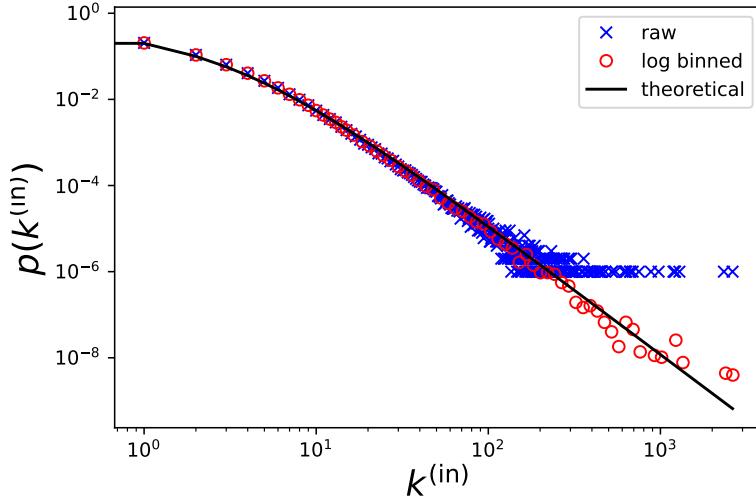


Figure 4.16: The degree distribution from a numerical simulations of the Price model with 10^5 nodes, $m = 2$ and $\mu = 2$. Each point represents the density found in one logarithmic bin (where each bin is roughly 1.1 times larger than the previous bin) for one run of a model at a given N value. The points are placed at $\sqrt{k_{\min}k_{\max}}$ where k_{\min} (k_{\max}) is the lowest (highest) value of $k^{(\text{in})}$ included in that bin. Even with this log binning, we can see finite size effects and binning artefacts at large degrees as the deviation of data points from the predicted theory becomes large. The continuous line represents the solution of form given in Equation 4.8.22.

for some constant A . This is easily checked by substituting (4.8.18) into (4.8.17) and using the property that $\Gamma(z + 1) = z\Gamma(z)$ (A.1).

In our case we see that (4.8.16) is in the form of (4.8.17) so we can use (4.8.18) to give us the solution²⁹

$$p(k^{(\text{in})}) = A \frac{\Gamma(k^{(\text{in})} + \mu)}{\Gamma(k^{(\text{in})} + \mu + 2 + (\mu/m))} \quad \text{for } k^{(\text{in})} \geq 0. \quad (4.8.19)$$

We still need to fix the normalisation constant A in (4.8.19). We could use the normalisation required for probability distributions but we will check that below when we look at the cumulative distribution $p_{\leq}(k^{(\text{in})})$ in (4.8.26). To find the constant A it is simpler to use the boundary condition that $p(k^{(\text{in})}) = 0$ for $k^{(\text{in})} < 0$ to see that the master equation (4.8.11) gives us

$$p(k^{(\text{in})} = 0) = \frac{m + \mu}{m + \mu + m\mu}. \quad (4.8.20)$$

Comparing this value (4.8.20) to the general solution (4.8.19) we see that

$$p(k^{(\text{in})} = 0) = A \frac{\Gamma(\mu)}{\Gamma(\mu + 2 + (\mu/m))} = \frac{m + \mu}{m + \mu + m\mu}. \quad (4.8.21)$$

and hence the full solution is

$$p(k^{(\text{in})}) = \frac{(m + \mu)}{(m + \mu + m\mu)} \frac{\Gamma(\mu + 2 + (\mu/m))}{\Gamma(\mu)} \frac{\Gamma(k^{(\text{in})} + \mu)}{\Gamma(k^{(\text{in})} + \mu + 2 + (\mu/m))} \quad (4.8.22)$$

for $k^{(\text{in})} \geq 0$.

²⁹Since the (4.8.16) is for any positive integer $k^{(\text{in})}$ values, when use the form (4.8.18) for $z = k^{(\text{in})} = 1$ the solution is also valid for $k^{(\text{in})} = 0$ since (4.8.17) links z to $(z - 1)$ values.

It is sometimes convenient to rewrite these ratios of Gamma functions in terms of BETA FUNCTIONS $B(x, y)$ (see (A.9))

$$B(x, y) = B(y, x) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}. \quad (4.8.23)$$

In terms of Beta functions we find that

$$p(k^{(\text{in})}) = \frac{B(k^{(\text{in})} + \mu, 2 + (\mu/m))}{B(\mu, 1 + (\mu/m))} \quad \text{for } k^{(\text{in})} \geq 0. \quad (4.8.24)$$

While this solution (4.8.22) is an exact solution to the difference equation with the boundary condition $p(k^{(\text{in})}) = 0$ for $k^{(\text{in})} < 0$, we should remember that the original equation is approximate and is strictly only for infinite time so this will only be an approximate analytic solution for the average degree distribution found at long times in numerical simulations. Nevertheless, it turns out to be a very good approximation in most regards.

The headline result for this analytic solution (4.8.22) to the Price model master equation is that for large $k^{(\text{in})}$ this solution behaves like a power law as Stirling's approximation (A.6) gives us that (see (A.7))

$$\lim_{k^{(\text{in})} \rightarrow \infty} p(k^{(\text{in})}) \propto \frac{1}{(k^{(\text{in})})^\gamma}, \quad \gamma = 2 + \frac{\mu}{m}. \quad (4.8.25)$$

A SCALE INVARIANT FUNCTION $f(x)$ (also called a homogeneous function) is one defined by the property $f(x/b) = C(b)f(x)$ for any x and b and for some function $C(b)$. The only functions which satisfy this property are power-law functions. Such functions have no inherent scale and lead to what is often called SCALE-FREE BEHAVIOUR.

In our case, we see this behaviour for large $k^{(\text{in})}$ in our Price model. So when we look at nodes with in-degree $k^{(\text{in})}$ the relative numbers of nodes of twice that in-degree, $2k^{(\text{in})}$ are the same independent of the scale $k^{(\text{in})}$, that is $n(2k^{(\text{in})})/n(k^{(\text{in})}) = (2)^{-\gamma}$ is independent of $k^{(\text{in})}$. Changing the size of the "ruler" in degree space, the size of $k^{(\text{in})}$ in the expression $n(2k^{(\text{in})})/n(k^{(\text{in})})$, does not alter the answer.

We also see that pure scale invariant behaviour is only seen in the theoretical infinite degree limit. This is common as in almost any real system there will be some scale. It is just that in some region, here for large degree, the effect of this scale becomes negligible. In the Price model, we can see clearly from the solution (4.8.22) that the power law only emerges when the degree is much bigger than μ and $(\mu + 2 + (\mu/m))$. So here we have μ acting as a scale for the degree. The existence of this small scale in the Price model also explains why we get a 'failure' of scale invariance. That is for small degree, close to the scale $k^{(\text{in})} \sim \mu$, we never expect to get a pure power law. We can interpret this as the effect of the small degree scale. It is also clear in the log scale plots of Fig. 4.16 where we see a reasonable straight line (power law behaviour) for large degree but clear curvature and deviation from power law at low degree. The message is that if you suspect you have a scale, perhaps at some extreme value for your data, it may well be worth trying to include its effects rather than forcing your data to fit a pure power law. Not only will you get better fits, you will reveal the size of the scale and make better measurements of the power in the power law region. For the Price model, the Yule distribution in (4.8.22) is the natural way to incorporate the small scale effects and should be the basis of any attempted fits to data.

One of the big advantages of the Price model is that it is solvable analytically under the same approximations that gave us the master equation (4.8.2). This means that we can obtain other properties of the model in a closed form. For instance the cumulative distribution $p_{\leq}(k^{(\text{in})})$ is

found to be (e.g. use the identity (A.11) on the Beta function in (4.8.24)) (T)³⁰

$$p_{\geq}(k^{(\text{in})}) = \sum_{n=k^{(\text{in})}}^{\infty} p(n) = \frac{B(k^{(\text{in})} + \mu, 1 + (\mu/m))}{B(\mu, 1 + (\mu/m))}, \quad (4.8.26)$$

$$p_{\leq}(k^{(\text{in})}) = \sum_{n=0}^{k^{(\text{in})}} p(n) = 1 - p_{\geq}(k^{(\text{in})} + 1). \quad (4.8.27)$$

This allows us to check that the normalisation is correct as we require $p_{\geq}(k^{(\text{in})} = 0) = \sum_{n=0}^{\infty} p(n) = 1$ which is clearly the case from (4.8.26).

4.8.4 Price Model Degree Evolution

We can also solve the Price model by looking at the evolution of the in-degree of any one node, one average. Suppose that on average the node added at time τ has in-degree $k^{(\text{in})}(t, \tau)$ at time t . Using the same approximations as used when creating the master equation we can estimate the evolution of $k^{(\text{in})}(t, \tau)$ to be

$$k^{(\text{in})}(t+1, \tau) = k^{(\text{in})}(t, \tau) + m\Pi(k^{(\text{in})}, t), \quad \text{for } t > \tau \quad (4.8.28)$$

with boundary condition $k^{(\text{in})}(t, \tau) = 0$ for all $t \leq \tau$. Substituting in the same form for the attachment probability Π , we have that

$$k^{(\text{in})}(t+1, \tau) = k^{(\text{in})}(t, \tau) + m\Pi(k^{(\text{in})}, t), \quad \text{for } t > \tau \quad (4.8.29)$$

If we use the same attachment probability Π of (4.8.3) then we have that

$$k_i^{(\text{in})}(t+1) = k_i^{(\text{in})}(t, \tau) + \frac{m(k^{(\text{in})}(t, \tau) + \mu)}{(m + \mu)t}. \quad (4.8.30)$$

It is useful to work in terms of the total degree of each node $k_i = (k_i^{(\text{in})}(t, \tau) + \mu)$ as then we have that

$$k_i^{(\text{in})}(t+1) + \mu = (k_i^{(\text{in})}(t, \tau) + \mu) \left(1 + \frac{m}{(m + \mu)t}\right). \quad (4.8.31)$$

To solve this in terms of Γ functions in terms of the variable t using (4.8.17), we need to make the last term of the form $(t+a)/(t+b)$ so we rewrite our equation as

$$\Rightarrow \frac{k_i^{(\text{in})}(t+1, \tau) + \mu}{k_i^{(\text{in})}(t, \tau) + \mu} = \frac{t + (m/(m + \mu))}{t} \quad (4.8.32)$$

So finally we find that the degree of node i at time t will be, since $k_i^{(\text{in})}(\tau, \tau) = 0$,

$$k_i^{(\text{in})}(t, \tau) + \mu = \mu \frac{\Gamma(\tau)}{\Gamma(\tau + (m/(m + \mu)))} \frac{\Gamma(t + (m/(m + \mu)))}{\Gamma(t)} \quad \text{for } t \geq \tau. \quad (4.8.33)$$

In the long-time limit³¹ we have that $k_i^{(\text{in})}(t, \tau)$ grows as $(t)^{\beta}$ where $\beta = m/(m + \mu) = 1/(\gamma - 1)$. We can see that $0 < \beta < 1$ so whatever the node is, however strong the cumulative advantage, the degree is growing sub-linearly. In particular, the node with the largest degree, the oldest

³⁰(T) Can we absorb the prefactor into the beta functions?

³¹In fact if $t \gg 1$ and $\tau \gg 1$ then we have that $k_i^{(\text{in})}(t, \tau) \approx \mu(t/\tau)^{\beta}$.

one with smallest τ is attached to an ever smaller fraction of the edges in the network. We can also see in this result (4.8.33) that since the initial graph has the oldest nodes, the smallest τ values, these have have the longest time to grow and so these are the largest degree nodes.

We can take this analysis one step further. We note that such solutions for the difference equation (4.8.28) for the time-evolution of the in-degree of a node require a boundary condition to set the initial value. Since $k_i^{(\text{in})} = 0$ for all new nodes, the initial attractiveness comes purely from the fitness factor μ . If we allow this fitness to vary³² from node to node, so μ_i is the fitness of node i , then provided the fitness parameter does not vary too much from a mean value $\bar{\mu}$, we can use μ_i in (4.8.28) except in the factors from the Π normalisation where we use $\bar{\mu}$. That is our evolution equation for $k_i^{(\text{in})}(t)$ becomes

$$k_i^{(\text{in})}(t+1) = k_i^{(\text{in})}(t, \tau) + \frac{m(k_i^{(\text{in})}(t, \tau) + \mu_i)}{(m + \bar{\mu})t} \quad (4.8.34)$$

The solution proceeds as before to give us that

$$k_i^{(\text{in})}(t, \tau) + \mu_i = \mu_i \frac{\Gamma(\tau)}{\Gamma(\tau + (m/(m + \bar{\mu}))} \frac{\Gamma(t + (m/(m + \bar{\mu})))}{\Gamma(t)} \quad \text{for } t \geq \tau. \quad (4.8.35)$$

What this shows is that the growth rate is the same for all nodes,

$$k_i^{(\text{in})}(t, \tau) \approx \mu_i \left(\frac{t}{\tau} \right)^\beta \quad \text{if } t \geq \tau \gg 1, \quad \text{where } \beta = \frac{m}{(m + \bar{\mu})}. \quad (4.8.36)$$

In the same way, small variations in the number of edges added to each new node will not make much difference and the m parameter in the growth represents the average number for edges added for each new node. So in these Price-like models the differences in the in-degree of nodes added to the network at similar times comes purely from the individual fitness of each node.

This result (4.8.35) also confirms what we mentioned before that the largest degree nodes are likely to be those nodes present in the initial graph since they will have the smallest values of τ . (T)³³

(T)*** THIS IS WHERE I GOT TO BA to Price update (end T)

4.8.5 Longest path in the Price model

Another example of how the Price model allows analytical solutions is to consider the length $\ell(t)$ of the longest path from any node t in the network to the first node at $t = 1$. For simplicity we imagine this node $t = 1$ to be the only sink node³⁴ of the DAG.

We can find the long-time behaviour of $\ell(t)$ using the following simple argument (Evans et al., 2020). We can view the attachment probability Π of (4.8.3) as made up of two parts: the $k_i^{(\text{in})}$ term which picks nodes using cumulative advantage, and the random attachment process μ term which picks nodes uniformly at random. So on average, the fraction of target nodes chosen with cumulative advantage will be $m/(m + \mu)$ since $\langle k^{(\text{in})} \rangle = m$, while the fraction of nodes chosen uniformly at random will be $\bar{p} = \mu/(m + \mu)$. The target nodes chosen with cumulative advantage are likely to be those created early in the process, those with large t as discussed in section 4.8.4, and these are unlikely to be relevant to a long path. Rather, the

³²For example, see the analysis of fitness variations in the Bianconi-Barabási model (Bianconi and Barabási, 2001).

³³(T)The paper of Guimarães et al. Guimarães Jr et al. (2005) appears to look at initial graph effects but only for very small graphs.

³⁴As long as the initial graph is small relative to t , then any of the nodes in the initial graph will give similar answer.

longest path from a node t is most likely to be made up of edges (t_n, t_{n+1}) which are close together say $t_n - t_{n+1} \ll t$. These shorter edges are therefore most likely to be created by the random attachment process.

So consider a node t . This node will on average have $m_{\text{rnd}} = m\mu/(m + \mu)$ edges created by random attachment. The probability that all these m_{rnd} randomly chosen source nodes are chosen between time 1 and time \hat{s} inclusive is $(\hat{s}/t)^{m_{\text{rnd}}}$. Suppose we consider the time $\hat{s}_{1/2}$ where with probability one half the time coordinate of the largest randomly chosen source node is $\hat{s}_{1/2}$ or less, then this sets the scale for the birth date of the youngest source node connected to node ($t = 1$), namely that $\hat{s}_{1/2} = \mu t$ where $\mu = 2^{-m_{\text{rnd}}}$. This is the previous node on the reverse greedy path from the initial node to node t . We can then estimate the numbers of steps it takes to get back to the source node at $t = 1$ as $\mu^\ell t \approx 1$ which leads to

$$\ell(t) = \frac{m\bar{p}}{\ln(2)} \ln(t). \quad (4.8.37)$$

The simplicity of the attachment probability in the Price model means we can also produce more detailed derivation using a mean-field approach which gives, as the leading order contribution for large times, the result

$$\lim_{t \rightarrow \infty} \ell(t) = m\bar{p} \ln(t) - m\bar{p}\psi(m\bar{p} + 1) + \sum_{n=2}^m \binom{m}{n} (-1)^{n-1} (\bar{p})^n \zeta(n) + O(t^{-1}) \quad (4.8.38)$$

where it is implicit that there is no contribution from the term with the sum for the case of $m = 1$. Here $\psi(z)$ is the digamma function and $\zeta(z)$ is the Riemann zeta-function. The details of the calculation are given in Appendix A.3 of the Supplementary Information.

4.8.6 Extending the Price model

See (Durrett, 2006) for mathematical summary.

(T)Undirected BA model. Variable fitness, variable m , adding edges between existing nodes, time-decay. **(end T)**

CHKNS model. Callaway et al. (2001) introduced the following simple version of a randomly grown graph. Start with $G_1 = \{1\}$ with no edges. At each time $n \geq 2$, we add one node and with probability δ add one edge between two randomly chosen nodes. Note that the newly added node is not necessarily an endpoint of the added edge and when n is large, it is likely not to be.

The Barabási-Albert model with $m = 1$ was introduced in a different context and under a different name by Szymanski (1987).

Price model: Summary

We have

- defined the Price model in terms of a growing network model and the principle of cumulative advantage or preferential attachment,
- given citations as an example where such a mechanism might be relevant
- set up the master equation for the model
- shown the long-time large-degree solution to be a power law with power $\gamma = 3$
- stated the properties of Generating Functions
- given the Price/BA master equation as a PDE for the generating function

4.9 Other Citation Models

One of the best examples of a real data set where a network representation has proved useful is that of the citations between documents. The nodes are naturally ordered since you can only cite older documents so the arrow of time provides a natural order for the nodes. In practice, there are cycles in the data because of errors or simply the ambiguity of a single publishing date for any document since they can appear in different versions or be assigned an official publishing date in several different ways. We will deal with methods of how to produce a DAG from data that only has a few cycles elsewhere (T)³⁵. The study of citation networks has a long history and one approach is to develop artificial models to help this work. Typically the models are trying to show how simple principles can lead to the patterns we observe in the data. There are a large number of such models and we have already seen one of the simplest, the Price model in section 4.8. The Price model produces a fat-tailed distribution similar to those seen in real data sets. However it has a single source node (or a small number in the initial group) whereas data on a large collection of papers has a large number of the oldest papers that are sources for the network. The Price network appears to be more like a snowball sample of a real set of papers. That is we start with one paper (or a small number published around the same time in the same topic) and then we follow the citations of that initial papers, and the citations of any papers citing the source papers, and so forth. If that structure is similar to the data you have, then the Price model may be of use.

However, most data sets and that includes most data on the citations between documents, has many sources of a similar age. Even if the number of documents in the data set increases exponentially as often happens on the scale of several years, the DAG representing the data looks like the single or few source DAG of the Price model. So here we will look at some examples of more sophisticated citation models that generate more interesting DAGs that might be of interest in a wider range of contexts.

(T)*** TO BE CONTINUED *** (end T)

³⁵(T)WHERE?

Chapter 5

Wild DAGs

Example and Data Sets.

- Citation Networks: Academic papers, patents, court judgements
- Food Webs
- Computer data sets
 - Programming Language
 - Computer Packages
 - Version control — git graphs (Courtiel and Pépin, 2024)
 - DAGs as blockchains Cryptocurrency
- Ratings and Rankings
- Bayesian Networks
- Spreadsheets
- Genealogy Trees
- Approximate DAGs

Analysis Tools. Here or in numerics?

- Geometric embedding of data. Effective Dimension of Data
- Sampling, snowball.
- Null models, configuration model.

Out in the wild forest of real data sets, there are many examples where DAGs are an extremely useful way to represent data. DAGs are good whenever data has a sense of order, perhaps coming from calendar time associated with each node or just set by an inherent sequence in the nodes representing steps in a process. In practice mistakes or limited accuracy in the recording of the data means that the data does not give you a perfect acyclic graph. Another problem is that there can be a range of different times linked to a node. If the raw data does lead to a network with cycles and if we want to work with a perfect DAG then section 6.6 gives several ways to extract a pure DAG from data. In this chapter we will focus on some of the different examples where DAGs have an important role.

5.1 Citation Networks

In a CITATION NETWORK (Price, 1965b, 1976) a node represents one document while the edges represent references, CITATIONS, made from one document to another document so there these edges have a natural direction, see Fig. 5.1. Examples include the citations found in academic papers and books (Price, 1965b, 1976, Moed et al., 1995), patents (Hall et al., 2001) (which must refer to relevant “prior art”, that is other recent relevant patents), or court judgements which build their case on previous legal judgements.

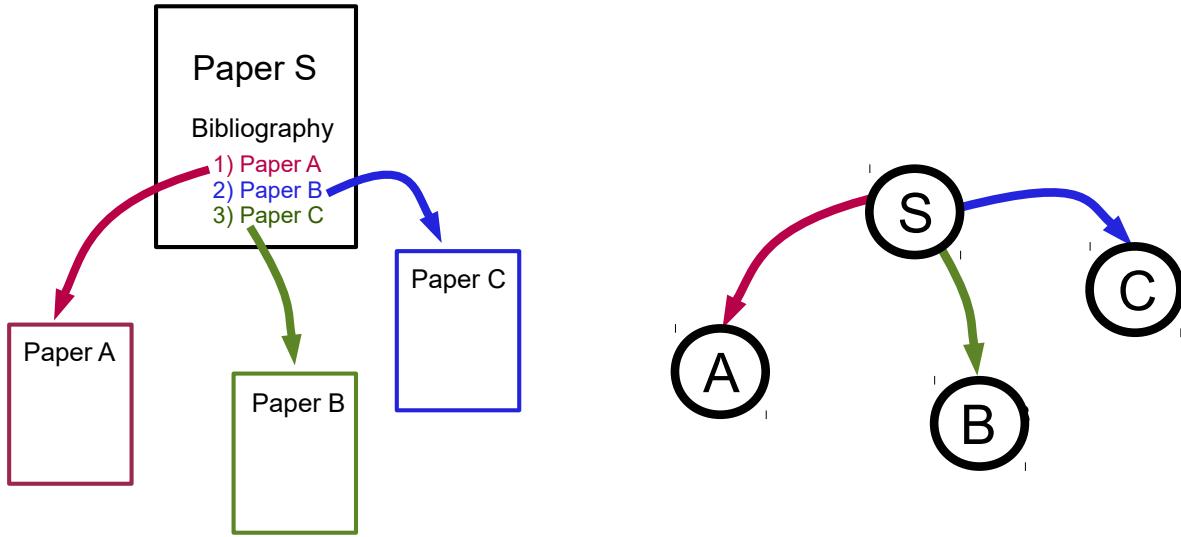


Figure 5.1: A sketch showing how a citation network is defined. On the left we illustrate citations made in the source document S to older documents A , B , and C . On the right is the corresponding network with nodes representing the corresponding documents. The edges are directed where we have chosen the direction to point away from the source document making the citation and so the edges point towards the older documents being cited.

One of the key observations about citation networks is that the tail of the degree distribution is generally found to be fat-tailed, often well summarised by a power law description or log-normal distributions, see Fig. 5.2.

A citation network can be used to analyse the impact that each publication has on the field or how a topic develops. The in-degree of each node is the citation count of that document. In academia that is often (mis)-used as a proxy for impact of a paper and for researchers of institutions producing papers e.g. see Moed et al. (1995). We can also use a citation network to identify the documents that represent key steps forward in a major scientific or technological breakthrough, for example see (Hummon and Doreian, 1989, Narin et al., 1997, Kalutkiewicz and Ehman, 2014, Barbieri et al., 2017, Kim et al., 2022, Alessandri, 2023), such as the development of a new vaccine (Ho et al., 2024b). Then the citation network can help show if these important stepping stones were held back by structural issues which we could address in the future, say in the way funding agencies work. We can see what earlier court judgements lay the foundations for recent legal rulings which might help develop work for a current legal case. We can use the topology of a network to define the distance or similarity between documents which might enable us to find related documents on a topic we are researching even if they are not directly cited by the documents we are already aware of. Given that such innovation is an evolving process, it is vital that this order over time in the documents is taken into account when we use the citation network in any analysis. Not surprisingly, these DAGs form the basis of several commercial systems which exploit the properties of the DAG in order to provide

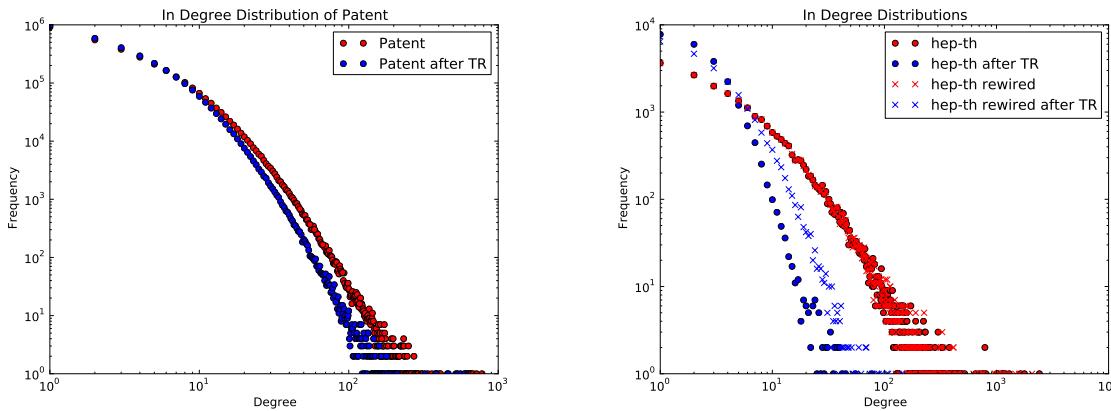


Figure 5.2: On the left the number of citations (the in-degree) for patents has a fat-tailed distribution ($N = 3.810^6$, $\langle k^{(\text{in})} \rangle = 12.8$). On the right the Zipf plot for the number of citations received by each paper (in-degree) against the rank of the paper (ranked by in-degree) for papers in the hep-th arXiv section ($N = 2.710^4$, $\langle k^{(\text{in})} \rangle = 8.8$). The corollary is that many academic papers (10% - 30%) have zero citations. In both cases we see that a few papers/patents get most of the citations. Figures from Clough et al. (2014) based on the 2003 KDD cup data (KDD cup, Gehrke et al., 2003).

customers with a better access to the key information they need amongst the vast numbers of documents out there. **(T)EXAMPLES (end T)**

One of the problems with data for any citation network is that documents rarely have a unique publication date. Different versions of a document often exist, reflecting different revisions or simply different stages of a publication process. For instance, an academic paper published in a journal has many dates associated with it (Haustein et al., 2015): there could be a date linked to the first submission to the journal, the final acceptance, the publication on the publisher's electronic web site, the official date on the published article, or even the date when the DOI number was issued. As a result, it is possible to get an entry in a bibliography which appears to be pointing forward in time. Section 6.6 gives some ways too extract a true DAG from such cases.

BING copilot

Source: Conversation with Bing, 08/05/2024 (1) CitNetExplorer - Analyzing citation patterns in scientific literature. <https://www.citnetexplorer.nl/>. (2) Is there a tool to visualize the academic citation network around a <https://academia.stackexchange.com/questions/86509/is-there-a-tool-to-visualize-the-academic-citation-network-around-a-specific-paper> (3) Citation Networks, SpringerLink. https://link.springer.com/chapter/10.1007/978-3-642-23068-4_7. (4) Analysis of citation networks as a new tool for scientific research. <https://link.springer.com/article/10.1557/mrs.2016.270>. (5) undefined.<http://www.annualreviews.org/>.

5.2 Scheduling

BING copilot A Directed Acyclic Graph (DAG) is a powerful tool for solving scheduling problems, particularly when tasks have dependencies that dictate the order in which they must be completed. In a DAG, nodes represent tasks, and edges indicate dependencies, meaning one task must precede another. This structure helps in identifying the most efficient sequence of operations, minimizing the overall completion time, and optimizing resource allocation.

Here's a simple example to illustrate how a DAG can be used in a scheduling problem:

****Example: Project Scheduling**** Imagine a project with five tasks: A, B, C, D, and E. The dependencies are as follows:

- Task B cannot start until Task A is complete.
- Task C cannot start until Task A is complete.
- Task D cannot start until both Tasks B and C are complete.
- Task E cannot start until Task D is complete.

This scenario can be represented as a DAG: In this DAG, the longest path from start to finish (A-C-D-E) determines the critical path, which is the minimum project duration. By scheduling tasks according to this graph, you ensure that all dependencies are met, and the project is completed in the shortest possible time.

****Advanced Scheduling with DAGs**** In more complex scenarios, such as those involving parallel processing or distributed systems, DAGs become even more useful. For instance, a paper on arXiv.org discusses a learning-based approach to scheduling DAG tasks(1). The algorithm employs reinforcement learning to iteratively add directed edges to the DAG, enforcing execution priorities and resource allocation for complex job nodes.

Another study uses a Deep Reinforcement Learning and Graph Convolution Network for a global DAG task scheduler(2). This model adapts to process complex interdependent task structures and minimizes the makespan of a DAG task.

Lastly, the Program Evaluation and Review Technique (PERT) is a practical application of DAGs for managing large projects, where the DAG represents the project and helps in finding the critical path(3).

These examples show how DAGs can be applied to various scheduling problems, from simple project management to complex computational task scheduling, providing a structured approach to optimize task completion times and resource usage.

Source: Conversation with Bing, 08/05/2024 (1) Learning to Schedule DAG Tasks - arXiv.org. <https://arxiv.org/pdf/2103.03412.pdf>. (2) A Global DAG Task Scheduler Using Deep Reinforcement Learning and Graph <https://rtcls.kku.github.io/webside/papers/IJ202111LCJ.pdf>. (3) Practical Applications of Directed Acyclic Graphs - Baeldung. <https://www.baeldung.com/cs/dag-applications>. (4) An ILP representation of a DAG scheduling problem. <https://link.springer.com/article/10.1007/s11241-021-09370-7>. (5) Cost-aware DAG scheduling algorithms for minimizing ... - Springer. <https://link.springer.com/article/10.1007/s11227-016-1637-7>.

5.2.1 Computer Datsets

- Computer Languages
- Computer Packages
- Version control — git graphs (Courtiel and Pépin, 2024)
- DAGs as blockchains Cryptocurrency

Programming Language DAG

This comes from the Supplementary Material 2 of Valverde and Solé (2015) who derived this from the English language pages on Wikipedia in May 2012 which describe programming languages¹.

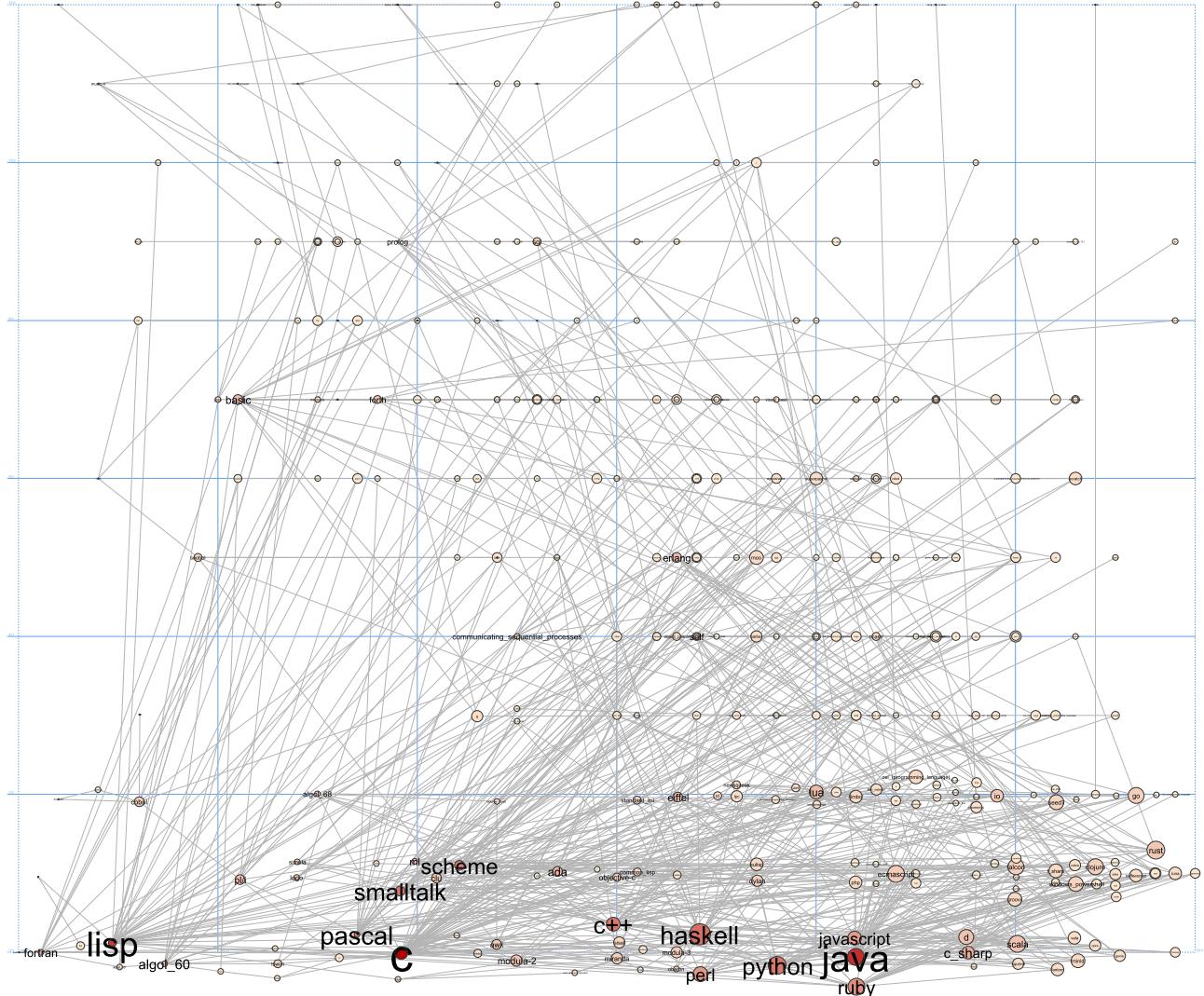


Figure 5.3: A historiograph of programming languages. Each node is a programming language with its own page found on the English language Wikipedia DAG in May 2012. Each node is a language with the size and colour of the node related to out-degree. The size of the label is related to the degree. For each node/language, any edges from earlier nodes are earlier languages which influenced the later language. The data comes from the supplementary material of Valverde and Solé (2015) but with a couple of corrections as discussed in a footnote in the main text. The horizontal coordinate of each node is set by the year the language was first introduced (earliest languages on the left) and the vertical coordinate is the criticality (3.2.1) with zero at the bottom. Nodes have been moved slightly (by hand) to avoid overlapping nodes in the visualisation. **(T)**Redo properly. **(end T)**

¹There were a couple of small issues with the data in the paper which have been corrected here. First the entry for *glulx* had a date of “????” in the original data which has been corrected to 1999 here. Second, there is a reciprocated edge from *dylan* to *eulisp* and back. Both had 1990 dates in the file.

name	year	In-degree	Out-degree	height	depth	criticality
speedcoding	1953	1	0	13	0	0
fortran	1954	13	1	12	1	0
fpl	1956	1	2	11	2	0
algol 58	1958	3	1	11	2	0
lisp	1958	39	2	10	3	0
algol 60	1960	15	1	10	3	0
cpl	1963	2	1	9	4	0
algol w	1966	2	1	9	4	0
iswim	1966	4	2	9	4	0
bcp1	1966	2	1	8	5	0
b	1969	1	2	7	6	0
pascal	1970	29	1	8	5	0
c	1972	50	5	6	7	0
sasl	1972	3	1	8	5	0
modula	1975	2	1	7	6	0
mesa	1976	2	2	7	6	0
awk	1977	6	3	5	8	0
modula-2	1978	7	4	6	7	0
kent recursive calculator	1981	2	1	7	6	0
c++	1983	22	6	5	8	0
miranda	1983	4	4	6	7	0
sisal	1983	2	4	5	8	0
modula-2+	1984	1	2	5	8	0
orwell	1984	1	1	5	8	0
modula-3	1986	4	4	4	9	0
oberon	1986	5	1	5	8	0
haskell	1987	16	14	4	9	0
perl	1987	14	7	4	9	0
python	1991	18	10	3	10	0
java	1995	35	9	3	10	0
javascript	1995	14	6	2	11	0
ruby	1995	14	9	2	11	0
c sharp	2001	11	5	2	11	0
d	2001	4	7	1	12	0
pizza	2001	2	1	2	11	0
scala	2003	2	9	1	12	0
squirrel	2003	1	4	1	12	0
fantom	2005	0	5	0	13	0
minid	2006	0	6	0	13	0
vala	2006	0	5	0	13	0
qore	2007	0	4	0	13	0
genie	2008	0	4	0	13	0
fortress	2011	0	3	0	13	0

Table 5.1: Information on the nodes on a longest path in the programming language DAG. Only nodes of centrality zero, i.e. all nodes that lie on a longest path, are listed.

Computer Package Dependency

Most software is built using pre-existing libraries which contain useful code that is used many times by completely different programmes. However this is true for these software libraries themselves. They may use other libraries to perform common tasks. The software

In computing, a package dependency occurs when a software package requires another package to function properly. This relationship is essential for the software's operation, as it relies on the external package's features or functionalities.

Here are some examples of package dependencies:

Direct Dependencies: These are the packages that your software directly calls upon. For instance, if you're developing a Python application, you might need the requests library to handle HTTP requests. You would list this library in your requirements.txt file, and it would be installed using a package manager like pip. Python

Transitive Dependencies: These are the packages that your direct dependencies rely on. Continuing with the Python example, if the requests library depends on urllib3, then urllib3 becomes a transitive dependency for your application. You might not interact with it directly, but it's necessary for requests to function. Python

Development Dependencies: These are packages required during the development process but not in the production environment. For example, a testing framework like pytest in Python would be a development dependency.

Package managers such as PIP for Python, Yarn or NPM for Node.js, and Maven for Java are tools that help manage these dependencies. They allow developers to install, update, and remove packages, as well as handle version control to avoid conflicts¹².

Managing dependencies is crucial because if a dependency is missing or incompatible, it can cause the software to break. Therefore, developers must carefully manage their software's dependencies to ensure smooth operation and maintenance¹².

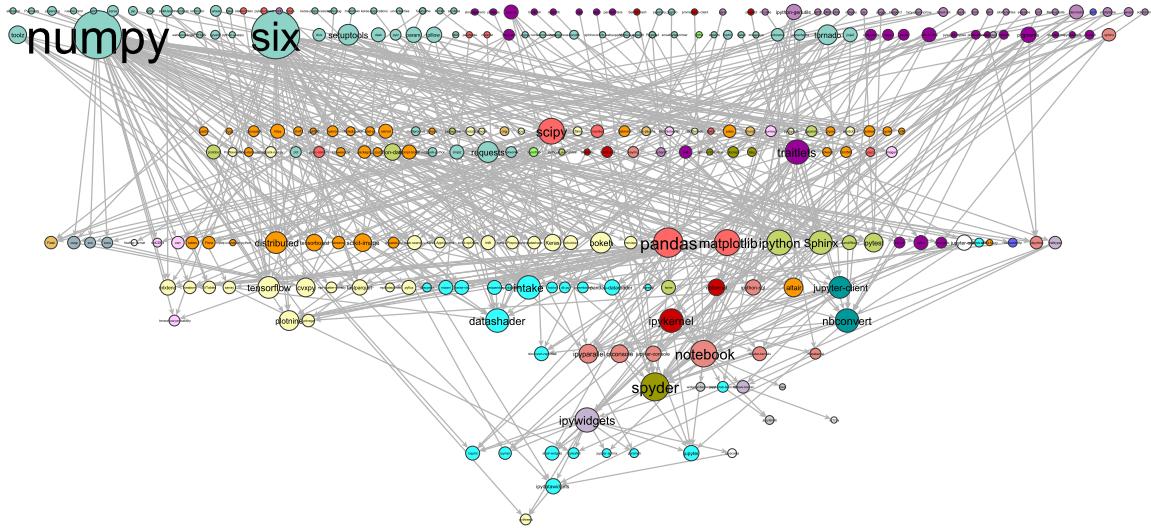


Figure 5.4: Python package dependency. Figure from Vasiliauskaite and Evans (2020)?

BING copilot A Directed Acyclic Graph (DAG) is an ideal structure for representing dependencies in computer packages because it can clearly illustrate which packages depend on others without creating cycles or loops that would indicate circular dependencies.

Here's a high-level explanation of how a DAG can be used for a computer package dependency problem:

1. **Nodes**: Each node in the DAG represents a package.
2. **Edges**: Directed edges between nodes represent dependencies, where an edge from package A to package B means that A depends on B.
3. **Acyclic**: The graph is acyclic, meaning it has no cycles, which ensures that there are no circular dependencies.
4. **Topological Ordering**: A DAG has at least one topological ordering, which is a sequence of the nodes such that for every directed edge from node A to node B, A comes before B in the ordering. This ordering can be used to determine the sequence in which packages should be installed or updated.

Example: Imagine you have three packages: 'A', 'B', and 'C'. Package 'A' depends on package 'B', and package 'B' depends on package 'C'. The DAG for this scenario would look like this:

In this DAG, 'C' would be installed first, followed by 'B', and finally 'A'. This ensures that each package's dependencies are met before the package itself is installed.

Real-World Example: Consider a more complex scenario where:

- Package 'A' depends on 'B' and 'C'.
- Package 'B' depends on 'C' and 'D'.
- Package 'C' has no dependencies.
- Package 'D' depends on 'E'.

The DAG would look like this:

The topological ordering could be ‘E’, ‘D’, ‘C’, ‘B’, ‘A’. This order respects the dependencies, ensuring that each package is installed after its dependencies.

Using a DAG to manage package dependencies helps avoid conflicts and ensures that all required packages are present before installing a package that depends on them. It’s a fundamental concept in package management systems like ‘apt’ for Debian-based systems, ‘yum’ for Red Hat-based systems, and more modern package managers like ‘npm’ for JavaScript or ‘pip’ for Python.

Source: Conversation with Bing, 08/05/2024 (1) Directed Acyclic graph in Compiler Design (with examples). <https://www.geeksforgeeks.org/directed-acyclic-graph-in-compiler-design-with-examples/>. (2) Directed Acyclic Graphs — DAGs — Examples — Gate Vidyalay. <https://www.gatevidyalay.com/directed-acyclic-graphs/>.

5.3 Food Webs

BING copilot A **food web** is a complex network of interconnected food chains within an ecosystem, illustrating the multiple pathways through which energy and nutrients flow as organisms feed on one another(1,2). It represents the various relationships between producers, consumers, and decomposers, showing how species are linked by their dietary preferences.

Examples of Food Webs - **Grassland Ecosystem**: Grass → Insect → Vole → Hawk; Grass → Insect → Frog → Fox; Grass → Insect → Vole → Fox(4). - **Marine Ecosystem**: Phytoplankton → Zooplankton → Small Fish → Larger Fish → Shark. - **Forest Ecosystem**: Leaves → Caterpillar → Bird → Hawk; Leaves → Deer → Wolf.

Representing a Food Web as a DAG A **Directed Acyclic Graph (DAG)** is a representation in which nodes denote entities, and edges represent the direction of energy flow, with the condition that there are no cycles to ensure that the energy transfer is unidirectional. In the context of a food web:

- **Nodes** represent the organisms (e.g., plants, herbivores, carnivores).
- **Edges** indicate who eats whom, pointing from the prey to the predator.
- **Levels** correspond to trophic levels, starting with producers at the bottom and apex predators at the top.

In a DAG representation of a food web, you would not find any organism that cycles back to be its own source of energy, which aligns with the real-world observation that energy flows in one direction through an ecosystem. This structure helps ecologists understand the complexity of ecological relationships and can be used to identify key species whose removal could disrupt the ecosystem balance.

Here's a simple textual representation of a food web as a DAG:

Plants (Producers)

Herbivores (Primary Consumers) - Omnivores (Secondary Consumers)

Carnivores (Tertiary Consumers) - Apex Predators (Quaternary Consumers)

In this DAG, arrows point from food sources to consumers, illustrating the flow of energy upwards through the trophic levels. The absence of cycles in this graph reflects the one-way flow of energy from producers to apex predators.

Source: Conversation with Bing, 08/05/2024 (1) What Is a Food Web? Definition, Types, and Examples - Treehugger. <https://www.treehugger.com/what-is-a-food-web-definition.html> (2) Food Web - National Geographic Society. <https://www.nationalgeographic.org/encyclopedia/food-web/>. (3) Food Chains - BBC Bitesize. <https://www.bbc.co.uk/bitesize/articles/zkwgvwx>. (4) Food Chains - BBC Bitesize. <https://www.bbc.co.uk/search?q=food+web+examples>. (5) Food chains and webs - Ecosystems and habitats - BBC. <https://www.bbc.co.uk/bitesize/articles/zjh4r2p>. (6) . <https://www.bing.com/search?q=food+web+as+a+DAG>. (7) Food Web: Concept and Applications — Learn Science at Scitable - Nature. <https://www.nature.com/scitable/knowledge/library/food-web-concept-and-applications-84077181/>. (8) undefined. <http://www.oxforddictionaries.com/>.

From copilot.

Mixed trophic impact (MTI) is a concept used in ecology to measure the relative impact of changes in the biomass of one component on other components within an ecosystem. This analysis helps in understanding both direct and indirect interactions among species, such as predation and competition(1,2).

The MTI method is based on an input-output approach, originally used in economic analysis, to assess how changes in one part of the system affect others(1). This can be particularly useful in managing fisheries and conserving ecosystems by predicting how alterations in one species' population might ripple through the food web(1).

Would you like to know more about how this concept is applied in real-world scenarios?

Source: Conversation with Copilot, 10/3/2024 (1) Trophodynamic indicators for an ecosystem approach to fisheries. <https://academic.oup.com/icesjms/article/62/3/430/661904>. (2) Predicting the sign of trophic effects: individual-based simulation <https://link.springer.com/10.1007/s00343-006-0068-1>. (3) Ulanowicz and C.J. Puccia, MIXED TROPHIC IMPACTS IN ECOSYSTEMS - JSTOR. <https://www.jstor.org/stable/pdf/43461017.pdf>. (4) Trophodynamic indicators for an ecosystem approach to fisheries P.M. Cury, L.J. Shannon, J-P. Roux, G.M. Daskalov, A. Jarre, C.L. Moloney, D. Pauly ICES Journal of Marine Science, Volume 62, Issue 3, 2005, Pages 430–442, <https://doi.org/10.1016/j.icesjms.2004.12.006>. <https://doi.org/10.1016/j.icesjms.2004.12.006>.

In a FOOD WEB or TROPHIC NETWORK, the nodes are species linked whenever we have a pair of species where one if the prey of the other predator species. We will use directed edges from prey u to predator v indicated by an entry in the adjacency matrix² of $A_{vu} = 1$. This means that the arrows in our diagrams point in the direction of the flow of energy in an ecosystem. It is this flow in energy that gives the order in these directed graphs and often leads food webs to be a DAG. In general the energy flows from plants up through herbivores to carnivores, and from smaller animals to larger animals. Cycles can occur but are rare. The simplest cycles are self-loops i.e. carnivorous activity. The next longest cycles come from reciprocated edges, where two species are both prey and predator to each other. This is typically hard to find in practice, for instance an individual carnivore will only be able to eat smaller carnivores so there is little

²The notation in ecology literature is often slightly different in that sometimes an entry $A_{uv} = 1$ in the adjacency matrix represents a directed edge from prey u to predator v . That means the matrices here are the transpose of those found in the literature with corresponding changes in the descriptions. The notation and descriptions used here are consistent with that used elsewhere in this book.

chance of reciprocity³. The most obvious cycles come from the behaviour of animals such as mosquitoes, ticks or leeches. While such links may be important in some contexts, they do obscure the basic hierarchy inherent in many ecosystems so if we want to study the hierarchy and understand its consequences, there may be good reasons to leave them out the odd link that creates a cycle.

Food webs are difficult to construct in practice. Trying to find out the food sources of *all* the organisms in an ecosystem is almost impossible. For instance in a forest we have a massive range of organisms from the microbial activity breaking down matter all the way up to the few large top predators. Inevitably, many studies have to focus on a limited range of scales, missing the smallest single cell organisms or unable to get good statistics on the largest less numerous species. Even deciding what is a node, what is a species, in a food web can be tricky. Some species may be rare, others can be closely related to each other so perhaps are better thought of as a single node in the food web. Equally, there can be advantages to putting related but distinct species into a single category e.g. grasses. Finally, actually measuring the links is very difficult as observing exactly what is being eaten by one species takes a lot of work. So even when data is available it is often noisy and limited. Despite that there are a number of food webs available from network data repositories or from the ecology papers cited in this section.

A wide range of measures are applied to food webs (for example see (Vasas and Jordán, 2006, Estrada, 2007, Fedor and Vasas, 2009)). One of the key questions that researchers ask in ecology is what is the impact of changes to one species on the system as a whole. This analysis has to include both direct and indirect effects so the network representation of the ecosystem as a food web is an excellent way to encode this information. That is they are trying to access the importance of a node which in this context measure the impact on the whole ecosystem if the population of one species changes. The idea is that if one species were to disappear would not just be on the immediate predator and prey species of the lost species. Rather the effects would spread out through the ecosystem through indirect relationships producing a what is called a trophic cascade. The food web is an excellent example of how a network can be used to capture such indirect relationships.

In network language, the aim is to identify the species with the most impact on the ecosystem, i.e. the most important nodes in the food web. So the measures used to examine species in food webs are often examples of node centrality measures for DAGs which may be of use in other contexts.

Typically the measures used in food webs work not just on DAGs but they will also function also on directed networks which are not strongly connected, i.e. there are sinks and sources. One reason is that these measures tend to look at how changes in one species can have positive and negative effects both along edges and in the reverse direction. As such these measures may be of use in other contexts. (**T**)Really true? Most look like DAG measures. Indicate in each case. (**end T**)

We will look at a few of these measures here:-

- An omnivory index, representing the trophic specialization of the predator (Christensen et al., 2008).
- The keystone index (K), as defined by Libralato et al. (2006) or is it Jordan et al. (1999).
- Mixed trophic impact (MTI) Ulanowicz and Puccia (1990).

³Of course pack/herd behaviour can compensate for the limitation imposed by the size of individual animals. Likewise, juveniles will always be smaller than adult forms so the age difference of predator and prey can also compensate for average size differences. These examples remind us that food webs are trying to encode a complicated set of complex relationships in ecology.

(T) Move these to the properties section as examples of generic centrality measures. We can rewrite in a more sensible way so we have it written in terms of diffusion from the right to left. (end T)

The KEYSTONE INDEX K_v (T)⁴ was defined by Jordan et al. (1999) (see also Harary (1961), Libralato et al. (2006), Vasas and Jordán (2006), Fedor and Vasas (2009)) but inspired by the work of Harary (1961) who drew on his ‘status-contrastatus’ measures to analyse hierarchies in organisations Harary (1959), see section 3.1.2. The keystone index emphasizes the relationship between a species and the ancestor and descendant species i.e. those lying in either its forward or backwards connection kernels. (T)⁵ Indirect relationships (T)⁶ with species where there is no path to the species v of interest, outside the connection kernels of a species v , do not contribute to these indices. The other key feature is that the keystone index is based on binary relationships and does not use any other information available for the food web such as the biomass or energy flowing between nodes. The definition of the keystone index is

$$K_v = K_v^{(b)} + K_v^{(t)} \quad (5.3.1)$$

$$K_v^{(b)} = \begin{cases} \sum_u \frac{1}{k_u^{(in)}} A_{uv} (1 + K_u^{(b)}) & \text{if } k_u^{(in)} > 0 \\ 0 & \text{if } k_u^{(in)} = 0 \end{cases} \quad (5.3.2)$$

$$K_v^{(t)} = \begin{cases} \sum_w \frac{1}{k_w^{(out)}} A_{vw} (1 + K_w^{(t)}) & \text{if } k_w^{(out)} > 0 \\ 0 & \text{if } k_w^{(out)} = 0 \end{cases} \quad (5.3.3)$$

where A_{vu} is one if species u is prey for predator v and zero otherwise. Note that the order of the indices on the adjacency matrix \mathbf{A} in these expressions changes. The first term, the bottom-up keystone index $K_v^{(b)}$, puts prey species u at the bottom of a network of predators, the ancestors in the forward connection kernel in our language. The bottom-up keystone index $K_v^{(b)}$ looks at each predator u of species v and considers the impact of predator u to be divided equally between all the prey species of u . The impact of species u all of its prey is deemed to be one (for the species u itself) plus the effective impact derived from predator species higher up the food web, i.e. $K_u^{(b)}$. The prefactor $1/k_u^{(in)}$ divides the impact of u equally between all its prey.

The second expression, $K_v^{(t)}$, captures the reverse relationship and looks at species v as a predator at the top of a subgraph, the set of descendants in the backwards connection kernel, so this expression looks down the food web.

We can write these indices in a simpler form⁷

$$K_v^{(b)} = \sum_u \left[\sum_{n=0}^{\infty} (\mathbf{T}^{(in)})^n \right]_{vu}, \quad T_{vu}^{(in)} = \begin{cases} \frac{1}{k_u^{(in)}} A_{uv} & \text{if } k_u^{(in)} > 0 \\ 0 & \text{if } k_u^{(in)} = 0 \end{cases}, \quad (5.3.4)$$

$$K_v^{(t)} = \sum_u \left[\sum_{n=0}^{\infty} (\mathbf{T}^{(out)})^n \right]_{vu}, \quad T_{vu}^{(out)} = \begin{cases} \frac{1}{k_u^{(out)}} A_{vu} & \text{if } k_u^{(out)} > 0 \\ 0 & \text{if } k_u^{(out)} = 0 \end{cases}. \quad (5.3.5)$$

Note how the flow of indirect effects for the bottom-up keystone measure $K_i^{(b)}$ is using the transpose of the adjacency matrix to reflect the fact this is looking in the opposite direction of the edges given our definitions. Equally, the top-down measure $K_i^{(t)}$ is in the opposite direction and follows the direction of the edges in the food web as defined here.

⁴(T) How does this differ from ‘keystonness’?

⁵(T) (vertical interactions)

⁶(T) (horizontal interactions)

⁷We cannot usually write these sums as $\sum_n (\mathbf{T})^n = [\mathbf{I} - \mathbf{T}]^{-1}$ because of sink and source nodes, or more generally, the lack of a strongly connected directed network. For a DAG, these sums over powers \mathbf{T}^n are zero for when n is larger than the height of the DAG, $n > H$.

(T) Best to reverse the direction of edges and then switch notation so in and out switch. Change A to W to indicate weighted graph. (end T)

The MIXED TROPHIC IMPACT (MTI) was defined by Ulanowicz and Puccia (1990) (taking inspiration from Leontief Input-Output models of macroeconomics) is another measure of who changes in one species in a food web will have on the whole ecology. That is it tries to quantify what effect a small change in one species has on the other species. Mixed trophic impact uses a weighted DAG where W_{vu} gives amount of energy⁸ derived from a prey species u by a predator species v , i.e. W_{vu} is the energy flowing from prey species u to a predator species v . The first step is to look at what fraction, (T)⁹ $T_{vu}^{(in)}$, of the input to predator v comes from prey species u .

$$T_{vu}^{(in)} = \begin{cases} \frac{1}{s_v^{(in)}} W_{vu} & \text{if } s_v^{(in)} > 0 \\ 0 & \text{if } s_v^{(in)} = 0 \end{cases} \quad \text{where } s_v^{(in)} = \sum_u W_{vu}, \quad (5.3.6)$$

This is a simple generalisation of the diffusion matrix $\mathbf{T}^{(in)}$ used in the keystone measure definitions from unweighted A to weighted W food web adjacency matrices.

In the same way we look at the effect of the predator v on prey u by finding the total energy provided by prey species u and we then consider the fraction $\mathbf{T}_{vu}^{(out)}$ of this that is consumed by predator species v . It basically means look at the energy provided by prey species u to all predators that consume u and we find the fraction (T)¹⁰ $T_{vu}^{(out)}$ of this energy goes to predator v . That is we define

$$T_{vu}^{(out)} = \begin{cases} \frac{1}{s_u^{(out)}} W_{vu} & \text{if } s_u^{(out)} > 0 \\ 0 & \text{if } s_u^{(out)} = 0 \end{cases} \quad \text{where } s_u^{(out)} = \sum_v W_{vu}. \quad (5.3.7)$$

Note that the entries of these matrices lie between 0 and 1, $0 \leq T_{vu}^{(in)}, T_{vu}^{(out)} \leq 1$. In particular, when the species v has both predators and prey (so it is not a sink or source node) then the v -th row of $\mathbf{T}^{(in)}$ and the v -th column of $\mathbf{T}^{(out)}$ sum to one. Conversely, for columns or rows associated with either a source or sink matrices, these are zero¹¹

We now ask what is the effect of a small ‘zeroth-order’ increase $\Delta_v(0)$ in the population of species v has on species u . We can look at two types of contribution.

First when u is prey for v as a predator, the first-order the impact on prey u is defined to be a positive amount $\Delta_u^+(1) = \Delta_v(0)T_{vu}^{(in)}$ since prey species u is only a fraction of the diet of predator v .

In the same way we can consider how the same zeroth-order increase $\Delta_v(0)$ in species v viewed as a prey of a predator species w . In this case the impact on the predator w will be negative and is defined to be $\Delta_w^-(1) = -T_{vw}^{(out)}\Delta_v(0)$.

Putting these two effects, consider the first-order impact $\Delta_v(1)$ on species v given some set of zeroth order changes in all the species $\{\Delta_u(0)\}$. We have to sum over all species u as both prey (giving positive changes $\Delta_u^+(1)$) and as predator (giving negative changes $\Delta_u^-(1)$). This gives us the total first-order change as

$$\Delta_v(1) = \left(\sum_u \Delta_u(0) T_{uv}^{(in)} \right) - \left(\sum_u T_{vu}^{(out)} \Delta_u(0) \right) = \sum_u \Delta_u(0) Q_{uv} \quad (5.3.8)$$

⁸Note that this method does not consider the flows from/to species from other sources e.g. respiratory output is not considered here. It is only the flow of energy that comes from predation of species represented by nodes that is included. Also note, our convention is that the edge represented by A_{vu} runs from prey u to predator v . The sense of direction is the same in the literature but the literature represents this by a matrix which the transpose of the A here. So care is need when comparing these expressions to those in the literature.

⁹(T)This is G_{ij} in the literature

¹⁰(T)This is F_{ij} in the literature

¹¹So these are not true stochastic matrices where every row (or every column) must sum to one. (T)Have we included this concept somewhere, otherwise remove it. (end T)

where we have defined the matrix \mathbf{Q} such that

$$Q_{uv} = T_{uv}^{(\text{in})} - T_{vu}^{(\text{out})} \quad (5.3.9)$$

$$= \begin{cases} \frac{1}{s_u^{(\text{in})}} W_{uv} - \frac{1}{s_u^{(\text{out})}} W_{vu} & \text{if } s_u^{(\text{in})}, s_u^{(\text{out})} > 0 \\ -\frac{1}{s_u^{(\text{out})}} W_{vu} & \text{if } s_u^{(\text{in})} = 0, s_u^{(\text{out})} > 0 \\ \frac{1}{s_u^{(\text{in})}} W_{uv} & \text{if } s_u^{(\text{in})} > 0, s_u^{(\text{out})} = 0 \\ 0 & \text{if } s_u^{(\text{in})} = s_u^{(\text{out})} = 0 \end{cases}. \quad (5.3.10)$$

This matrix has a few interesting properties. First for any row u where node u has non-zero $s_u^{(\text{in})}$ and $s_u^{(\text{out})}$, i.e., where the node u is neither a source or a sink, the corresponding row sum is zero, so $\sum_v Q_{uv} = 0$. Otherwise the sum across entries in row u is $+1$ if node u is a sink, -1 if node u is a source¹². In general the entries all lie between -1 and 1 , $-1 \leq Q_{ji} \leq 1$ so that means that $(\mathbb{1} - \mathbf{Q})$ is a non-negative matrix.

The first order effects $\Delta(1)$ in turn lead to second order effects $\Delta(2) = \Delta(1)\mathbf{Q} = \Delta(0)\mathbf{Q}^2$ and so forth. In general the n -th order effect is $\Delta(n) = \Delta(0)\mathbf{Q}^n$ in terms of the original perturbation. Of course, ecological systems are highly complicated non-linear systems so the use of these linear equations to capture the changes is really some sort of small perturbation approximation. Nevertheless, it provides a simple way to get some insight into the stability of the species given their interactions encoded in the food web.

The measure favoured in ecology is to sum up these changes to all orders. That is the stability measure S_u for species u is given by the sum of the changes at all order induced by the initial zero-order changes $\Delta(0)$, so

$$S_u = \sum_{n=1}^{\infty} \Delta_u(n) = \sum_{n=1}^{\infty} \sum_v \Delta_v(0)[\mathbf{Q}^n]_{vu} \quad (5.3.11)$$

$$= \sum_v \Delta_v(0)[\mathbf{M}]_{vu} \quad (5.3.12)$$

where we can show that we may write the mixed trophic impact matrix \mathbf{M} as

$$\mathbf{M} = (\mathbb{1} - \mathbf{Q})^{-1} - \mathbb{1}. \quad (5.3.13)$$

The final step is to say that, a priori, that to get a measure of the importance of each species to the system, we consider a case where all species are perturbed by the same amount at the zero-th level, so $\Delta_v(0) = \Delta(0)$, and we ask what is the fractional change in S_u for each species u , i.e. we look at $S_u/\Delta(0) = \sum_v M_{vu}$ the column sums of the mixed trophic impact matrix given our notation here.

In fact we can also see a connection between the keystone measure and the mixed trophic index. Both are written in terms of two diffusion matrices, $\mathbf{T}^{(\text{in})}$ and $\mathbf{T}^{(\text{out})}$ with the weighted network versions used for MTI in (5.3.6) and (5.3.7) reducing to the unweighted versions in (5.3.4) (5.3.5) if we set the weight adjacency matrix \mathbf{W} equal to the unweighted version \mathbf{A} . Then we note that the \mathbf{M} matrix used to define MTI is also

$$S_u = [\sum_v \sum_n \mathbf{Q}^n]_{vu} = [\sum_v \sum_n [\mathbf{T}^{(\text{out})} - \mathbf{T}^{(\text{in})}]^n]_{vu} \quad (5.3.14)$$

$$= [\sum_v \sum_n [\mathbf{T}^{(\text{out})}]^n]_{vu} + \dots + [\sum_u \sum_n (-1)^n [\mathbf{T}^{(\text{in})}]^n]_{vu} \quad (5.3.15)$$

$$= K_v^{(\text{t})} + \dots + [\sum_u \sum_n (-1)^n [\mathbf{T}^{(\text{in})}]^n]_{vu} \quad (5.3.16)$$

¹²Row sums are zero if it is for a completely disconnected node but we are assuming for simplicity that we are working with weakly connected networks where such nodes do not exist except for the trivial food web of just one node.

So the first term in the \mathbf{M} used for MTI is just the $K_v^{(t)}$ keystone measure. The last term is almost $K_v^{(b)}$ except that there is a sign difference between odd and even powers of $\mathbf{T}^{(in)}$. (T)
 ??? THE SIGN ISSUE HERE, ROW SUM/INDEX? (end T)

5.4 Ratings and Rankings

Suppose we have a group of objects which have been rated (an integer or real valued measure) or ranked (an ordering) by various different measures. This could be movies rated by different critics, academic papers whose impact has been measured in various ways, sports teams ranked by different organisations, or various environmental measures of different rivers. If there is a single measure used, then there will be a clear order for the objects. However if two or more measures are used working out the best film, the most polluted river, is not trivial and in fact there are many well developed ways to do this though never a perfect approach, see (Langville and Meyer, 2011).

We can also use a network approach to this problem. If we think of the objects as the nodes in our network, we can make a directed link link say from any object with a high rating to any object with a lower rating. If we do this for each type of rating, we have a multiedge a multilayer network and it is quite possible to use network analysis tools on this. However, if the goal is to produce a hierarchy, to find the best journal for my next academic paper, to prioritise the rivers with the worst environmental problem for cleanup work, then it is inevitable that at the end of the process we want at least a partial order between our objects. That is we want a network of bilateral comparisons between objects where there are no cycles, we want a representation of a DAG. Again, we will discuss how to remove cycles from arbitrary directed networks later in section ?? but here we will consider one of the most obvious ways to produce a poset, a DAG. That is we will only consider one object u to be more highly rated than another v if *each* rating of u is higher than the corresponding rating of v . That is if u_i is the rating of i under the i -th measure, then we have $u \succ v$ iff $u_i > v_i$ for all i . That is we use the cube space rule (C.1).

One area where this type of construction has been used is when the values given to each node are a set of different measurements or ratings for the object associated with the DAG and this is illustrated in Fig. 5.5. For example in Raveh and Landau (1993), the nodes are fifty states of the USA and the coordinates v_i are seven crime statistics for that state, given in terms of an integer rank, say 1 for the lowest numbers and up to 6 for the worst crime statistics. Another example where this method was used is for environmental studies. Each node v might represent a particular place on a river and the v_i represent the concentrations of different chemicals measured at that point Brüggemann et al. (1994a,b), Münzer et al. (1994), Brüggemann et al. (2001). As a final example this has been used to combine the ratings given to different academic journals.

BING copilot A Directed Acyclic Graph (DAG) is a powerful way to represent ratings and rankings of objects. In a DAG, objects are represented as nodes (or nodes), and the edges (or arrows) between them represent the direction of preference or ranking. Here's how it can be conceptualized:

- **nodes**: Each node in the DAG represents an object being rated or ranked.
- **Edges**: An edge from node A to node B indicates that object A is rated or ranked higher than object B.
- **Acyclic**: The graph is acyclic, meaning it has no cycles; you cannot start at one node and follow a sequence of edges that eventually loops back to the starting node.
- **Directed**: All edges are directed and point from higher-ranked objects to lower-ranked ones.

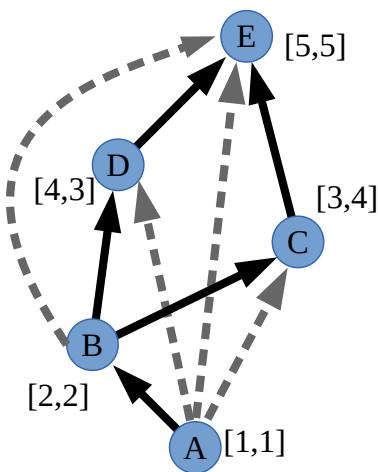


Figure 5.5: An example of how a cube space criteria may be used to combine different rankings of objects to produce a cube-space DAG and then the Hasse diagram of this DAG can give a simple visualisation of the relative importance of different objects. Every node is ranked using two different measures and these are given in the square brackets. Note there is no edge between nodes C and D because each has a higher rank than the other by at least one measure. All the edges, both solid and dashed, are present in the DAG which is transitively complete. However, we would normally just use the Hasse diagram visualisation where the dashed lines would be left out and just the transitively reduced graph would be shown.

This structure allows for the representation of complex relationships between objects, including partial orders and ties, where some objects may not be directly comparable. For example, if object A is ranked higher than object B , and object B is ranked higher than object C , then in the DAG, there would be a path from A to C passing through B .

In the context of preference data, DAGs offer a more general abstraction than simple lists of rankings. They can accommodate partial rankings and top- k lists, where only a subset of objects is ordered, and the rest are unspecified. This is particularly useful in situations where not all objects are compared against each other, or some preferences are unknown.

Moreover, DAGs can be used to aggregate and cluster sets of rankings from different sources. By comparing DAGs, one can measure the distance between different sets of preferences, which is useful for applications like meta-search engines or segmenting user bases in market research.

The concept of using DAGs to represent rankings extends beyond simple orderings to encompass a wide range of applications, including information retrieval, recommendation systems, and social network analysis(1,2).

In summary, DAGs provide a flexible and robust framework for representing and analyzing rankings and ratings of objects, accommodating the complexities and nuances of real-world preference data.

Source: Conversation with Bing, 08/05/2024 (1,3) Beyond rankings: comparing directed acyclic graphs. <https://link.springer.com/article/10.1007/s10618-015-0406-1> (Malmi et al., 2015) <https://users.ics.aalto.fi/ntatti/papers/malmi15dag.pdf>. (2) An Introduction to Directed Acyclic Graphs ggdag. <https://r-causal.github.io/ggdag/articles/intro-to-dags.html>.

5.5 Bayesian Networks and Causal Inference

One of the core tasks in science is to understand cause and effect. In fact, we would like to have a quantitative estimate of the effect coming from any given cause. To what extent does air pollution effect cancer rates? How does the gender of visitors to a web site alter the number of sales made? Do the heritage of a judge and defendant influence outcomes in court? At the

simplest level, we might look at correlations between some cause and the effect we are interested in. So are cancer rates higher in areas with high levels of air pollution? However, a correlation does not always mean there is a causal effect. Perhaps the people living in dense urban areas in a study are recent immigrants and there is something distinctive in their genetic makeup that is related to increased cancer risk.

The formal study of these questions is a topic known as CAUSAL INFERENCE (Pearl, 1995, Greenland et al., 1999, Morgan and Winship, 2015, Cunningham, 2021). What we are trying to do is to write down a formal statistical model in which if we have a value for every relevant cause, we can give a definitive probability distribution for the effect of interest. To investigate this idea, one of the best known methods is linear regression (see (Cunningham, 2021) for a simple introduction). In this case we assume that if we measured a value E for some effect (e.g. someone has cancer) given the values C_i of the n variables measuring possible causes, e.g. income, age, heritage, then this is related through some linear equation

$$E = \beta_0 + \sum_i \beta_i C_i + \xi \quad (5.5.1)$$

where ξ is some random variable. If we assume that the expectation value of this random variable is zero and that this is independent of the values of the causes x_i , then we can show that a least squares analysis can produce estimates $\hat{\beta}_i$ for each of the β_i parameters. All we need is a lot of data where for each example we have a value for E and for all the C_i variables, then we can produce estimates for $\hat{\beta}_i$ (in terms of some matrix of covariance values).

However, the linear regression model (5.5.1) is a very strong assumption, one that may not always be justified. These days we are able to handle much more general statistical models linking many variables. This is one place where causal inference becomes a useful tool to represent the probability $p(E|C_1, C_2, \dots, C_n)$ that we measure E in the effect (e.g. the probability someone has cancer) given the values C_i of the n causes (e.g. genetic heritage, age, gender). Two aspects of causal inference are immediately worth noting.

First, the variables which are not included in our probability $p(E|C_1, C_2, \dots, C_n)$ are just as important as those variables C_i identified as causes. We are making very definite statements about our model of the effect in those variables we leave out and by writing these down clearly we help later researchers to challenge and check these basic assumptions. For instance, perhaps the effect of the genetic makeup on cancer rates was not part of one study so if that is clear to readers of that study then other researchers can follow up that question and improve the model. So another key part of causal inference using DAGs is that it allows us to represent what variables we are considering in our analysis and the relationships between them.

The second point is that the properties of probabilities used in Bayesian analysis do not have a direction. That is the probability $p(A, B)$ that we measure A and B for two different variables (the joint probability) can be written in terms of the conditional probability $p(A|B)$, that is the probability that we measure A for the first variable if we know we have measured B for the second. If the probability that we measure B for the second variable $p(B)$ is known (the marginal probability of B) then we would use the relation that $p(A, B) = p(A|B)p(B)$. We could interpret this expression as variable B is a cause of variable A which is the effect we are trying to measure. However, algebraically, there is nothing special about which we call the first variable and which we call the second variable. So we could also work in terms of A causing B using $p(A, B) = p(B|A)p(A)$. In terms of the algebra it makes no difference how we write $p(A, B)$.

However, in real problems we want to express things in outcomes in terms of causes. So we are adding an order, a direction to what is otherwise a set of probability relations which in an abstract sense have no preferred direction or order. Again, the researchers defining their statistical model for a problem is imposing this order and this is an important part of their

model. For instance, if variable A is whether someone has cancer and variable B is if that person smokes, we are going to be interested in B as a cause and trying to infer how this effects the outcome A , if someone has cancer. We could try to create a model the other way round, trying to see if those with cancer are smokers. This reverse problem might even be of interest if, say, we have limited resources to help people quit smoking so we might want to know if targeting smokers is really as effective as our intuition might suggest. So assigning an order, a direction, to encode the cause and effect that we are modelling in our system is another part of causal inference and it is another reason why DAGs are the appropriate tool.

The use of a DAG to represent such cause and effect has become an important tool in understanding complex relationships. The DAG itself does not represent all the details, such as if variables are binary or continuous, the precise shapes of the probability distributions. One of the most important things it does is give a clear visual representation of the variables involved and their relationships or lack of relationship. Whole textbooks are dedicated to this use of DAGS to encode causal inference so we will just give the barest outline here.

A BAYESIAN NETWORK, also known as a belief network, represents the relationships between a set of random variables. Each node v represents a random variable, say x_v , and a probability distribution for that variable p_v . The probability distribution depends on the value of any preceding neighbours, that is the values of nodes u which are linked by edges (u, v) to node v . This means the probability distribution for x_v is in general a conditional probability in, with $p_v = p_v(x_v | \{x_u | (u, v) \in \mathcal{E}\})$. So the directed edges represent direct dependencies between variables.

This network must be acyclic to give a well defined model for our problem so that we can extract meaningful results. That is we have identified causes, whose variables are linked to source nodes, and we are trying to understand how these causes effect the outcomes, the variables linked to sink nodes.

If problems in the real world were as simple as this, we have a set of known causes $\{C_i\}$, and we know how the influence some effect whose value E we are trying to find, then we would not need a graphical method, it is just basic probability analysis. However, problems such as cancer rates have a huge number of variables that may, or may not, be important. In almost any problem we also have many unknown variables, either because we have no data on that variable or because we do not even know that such a variable is important to our study.

We will confine ourselves to the basic ideas of causal inference and show how they are represented by a DAG.

The first step is to define the nodes and edges in the DAG. In causal inference, every node represents a random variable. This could be something we measure, some will be viewed as causes, some as effects. Some variables may represent aspects that we don't have measurements of. So one of the goals of causal inference is to identify these variables, to decide if they are important to what we want to measure and, if they are, help us to plan how to deal with this problem.

The edges in a causal inference DAG represent dependence with an edge (x, y) , often written as $x \rightarrow y$ in causal inference texts, from a node representing variable x to a node representing variable y tells us that the probability for y in our model depends on the variable x . That is we have a statistical model that includes the conditional probability $p(y|x, \dots)$. For example, x could represent the educational level of parents in a family and y could represent the educational level of children and we might be asking what effect does the education of parents have on the chances that children go to university. Every incoming edge to the node y tells us about further random variables, x_1, x_2, \dots , upon which y depends. Our data might include information on some of these factors, such as cultural background, income, age of parents and so forth, or our data may have missed some of them.

The real interest in any network is when we look beyond single edges. So let us consider

what various triangles in a causal inference DAG mean. These represent the introduction of a third variable w , a third node, connected to x and y . What are the various ways that this third variable w can alter the way we try to get results for the variable we wish to measure, y ?

Consider the first example in Fig. 5.6. This represents a case where the value of a third variable w changes y , our variable of interest, both directly and indirectly via x . So if we are trying to understand the direct effect of x on y we will be unable to do that, we will be confounded, if we know not also know the value of w . For this reason causal inference calls the variable w in Fig. 5.6.i a CONFOUNDER. The relation from x to y via w is often written as $x \rightarrow w \leftarrow y$.

The edges connecting w to x and y are reversed in Fig. 5.6.ii compared to our first example. They tell us that the value of w does not effect either x or y , rather it is the other way round. So in principle it seems we can forget about w , just measure x and y to understand their relationship, our goal here. However, there is a danger here, one captured by the diagram Fig. 5.6.ii. What if, for some reason our data is conditioned on w ? Perhaps x represent the exam results of students entering university while and y represents their final university degree. Then w might be a student's income in the first ten years after their degree. The danger here is that we may have data that is conditioned on w , perhaps our records are incomplete but are only poor for low salary students. Now this would create a bias in the data and skew the results. A variable such as w is not on a path (as we defined in section 2.4) from the causes we are interested in, here x , to effects we are trying to measure, here y . Variables like w are called COLLIDER variables. The relation from x to y via w is often written as $x \leftarrow w \rightarrow y$. They appear harmless and can be ignored but there are many cases where they play a role in our data and influence results. The DAG helps us identify such collider variables so that we can make sure they are dealt with correctly.

A word of warning. The causal inference literature, like every field, introduces a lot of terminology but some network terms are defined differently from the ones we have used. In particular, a “path” in causal inference is one where the connection between successive nodes on the path do *not* have to be edges as long as there is an edge at one of the two possible directions. So for the two examples Fig. 5.6, there is no path (x, w, y) on the DAG as we have defined but there is such a path on the equivalent undirected network and this is how a “path” is defined in causal inference. We will stick with our definitions of a path¹³.

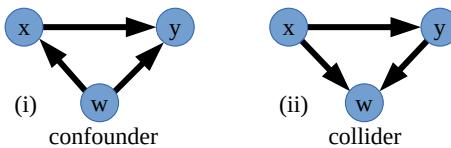


Figure 5.6: Two examples of how DAGs are used in causal inference. Here we are interested in understanding how the variable y depends on variable x . There is a direct dependence represented by the edge from x to y . However there is a third variable, w that is related to both x and y . In (i), some of the dependence of y on x is not direct but comes through the influence of x on w which in turn directly effects the value of y . Here w is called a “confounder”. In (ii), the edges connected to w are reversed and it is known as a “collider”.

Here is a simple example to illustrate a Bayesian network:

¹³Sometimes, the causal inference literature talks about a “causal path” which is one where successive nodes in a path are connected by a connected edge in the correct direction, i.e. this is what we would call a path in these DAGs. Then an acausal path is all the sequences that causal inference calls a “path” but where at least one of the successive node pairs is an edge in the wrong direction. In other fields, sequences of nodes on a directed network where successive nodes can be an edge in either direction are called semiwalks and this seems useful way to talk about such sequences on a directed network such as a DAG.

Imagine we have two events: Rain (R) and Sprinkler (S). We want to know the probability of the Grass being Wet (W). The Bayesian network would consist of three nodes, R, S, and W, with directed edges from R and S to W. The edges represent the fact that the wetness of the grass is dependent on whether it rained and whether the sprinkler was on.

The network would be accompanied by a set of probability tables. For instance:

$P(R)$ - the probability of it raining. $P(S)$ - the probability of the sprinkler being on. $P(W|R,S)$ - the probability of the grass being wet given the status of rain and sprinkler. Using this network, we can perform various inferences, such as calculating the probability of it having rained given that the grass is wet, which is a classic example of using Bayesian networks for reasoning under uncertainty.

In practice, Bayesian networks are used in various fields, such as:

Medicine: For diagnosing diseases based on symptoms¹. Bioinformatics: To understand the relationship between traits and genes¹. Speech Recognition: To model the relationship between utterances and time¹. These networks can be constructed by domain experts or learned from data, and once built, they can be used to perform inference, making them powerful tools for scenarios involving uncertainty and probabilistic reasoning.

BING copilot A **Bayesian network**, also known as a **belief network** or **Bayes network**, is a probabilistic graphical model that represents a set of variables and their conditional dependencies through a directed acyclic graph (DAG). Here's a breakdown of its components:

- **Nodes**: Each node represents a random variable, which can be discrete or continuous.
- **Edges**: Directed edges between nodes represent conditional dependencies.
- **Conditional Probability Tables (CPTs)**: Each node has an associated CPT that quantifies the effects of the parents on the node.

The network encodes joint probability distribution and can be used to compute the probability of outcomes given some evidence. It's particularly useful in scenarios where you need to make decisions under uncertainty and is widely used in various fields such as machine learning, diagnosis, and natural language processing.

Mathematically, for a set of variables

$$X = \{X_1, X_2, \dots, X_n\}$$

, the joint probability distribution can be factored into a product of conditional probabilities:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | Parents(X_i))$$

where

$$Parents(X_i)$$

are the parents of

$$X_i$$

in the graph. This factorization allows for efficient representation and computation of complex probability distributions.

(TSE from spreadsheet request.) Also see (2,4) Use of directed acyclic graphs (DAGs) to identify confounders in <https://academic.oup.com/ije/article/50/2/620/6012812> (Tennant et al., 2020). <https://doi.org/10.1093/ije/dyaa213>.

(From <https://dagshub.com/blog/an-introduction-to-directed-acyclic-graphs-dags-for-Biology>)

Biology DAGs exist in epidemiology to detect confounders. These are "unexpected variables" that can affect a study.

The structure of a DAG allows the person studying it to use it as a visual aid. This allows them to have easier discussions about underlying relations between possible causes. This is especially true for issues that have quite complex variables associated with them.

Retail Another area using DAGs to help grow their industry is the retail space. Retail, as well as other industries, are starting to switch toward a concept known as "customer journey marketing."

The idea is that nobody makes an instant decision to buy something. There is a "journey" the customer must be walked through. Retailers use advertising, and introduce their product, at multiple points throughout the journey. With the hopes of ultimately getting their prospect to buy.

Retailers use DAGs to visualize these journey maps, and decide what to focus on in order to improve their business.

Cryptocurrency/Blockchain Cryptocurrencies are all the rage these days. Elon Musk loves to tweet about them and get them to the moon. You probably heard that these coins rely on something called the blockchain.

This blockchain is defined by something called a Merkle Tree, which is a type of DAG. This means that DAGs are also responsible for one of the biggest shifts in the finance industry.

5.6 Spreadsheets

BING copilot In a spreadsheet, formulae can be represented as a Directed Acyclic Graph (DAG) to illustrate the dependencies between cells. Here's how it works:

- **Nodes**: Each cell in the spreadsheet that contains a formula is represented as a node in the DAG.
- **Edges**: If a formula in one cell (say, Cell A) references another cell (say, Cell B), then there is a directed edge from Cell B to Cell A. This indicates that Cell A is dependent on the value of Cell B.
- **Acyclic**: The graph is acyclic, meaning it has no cycles. This is crucial because a cycle would imply circular references, where cells depend on each other in a loop, leading to undefined values.

When a cell's value changes, the spreadsheet software uses the DAG to determine which other cells need to be updated. It does this by performing a topological sort on the DAG to find an order in which to recalculate the cells so that each cell is updated after all the cells it depends on have been updated. This ensures that all dependencies are resolved correctly and the spreadsheet remains consistent.

For example, if Cell C depends on Cell B, and Cell B depends on Cell A, the DAG will ensure that Cell A is recalculated first, then Cell B, and finally Cell C. This dependency graph must be a DAG; otherwise, the spreadsheet would not be able to resolve the calculations properly(1).

This representation allows for efficient recalculation of cell values when the spreadsheet is modified, as only the cells that are affected by the change need to be updated, rather than recalculating the entire sheet. It also helps in detecting errors

such as circular references, which are not allowed in spreadsheets because they make it impossible to calculate a definitive value for the cells involved(1).

Source: Conversation with Bing, 08/05/2024 (1) When to use DAG (Directed Acyclic Graph) in programming?. <https://softwareengineering.stackexchange.com/questions/171671/when-to-use-dag-directed-acyclic-graph-in-programming>. (2) An Introduction to Directed Acyclic Graphs (DAGs) for Data Scientists. <https://dagshub.com/blog/an-introduction-to-directed-acyclic-graphs-dags-for-data-scientists> (3) Directed Acyclic graph in Compiler Design (with examples). <https://www.geeksforgeeks.org/directed-acyclic-graph-in-compiler-design-with-examples/>. (4) undefined. <https://DAGsHub.com>. (5) en.wikipedia.org. https://en.wikipedia.org/wiki/Directed_acyclic_graph.

The DAG for a spreadsheet with formulae is defined as

$$G = (V, E)$$

, where

$$V$$

is the set of nodes and

$$E$$

is the set of edges. The adjacency list or matrix of

$$G$$

represents the spreadsheet's structure, where each entry

$$a_{ij}$$

indicates whether there's a direct dependency from cell

$$j$$

to cell

$$i$$

For a published paper that discusses the representation of spreadsheet formulae as DAGs, you can refer to the paper titled "Efficient and Compact Spreadsheet Formula Graphs" which explores the use of DAGs in the context of spreadsheets(2). This paper provides insights into how DAGs can be leveraged for efficient querying and maintenance of spreadsheet formula graphs, emphasizing the importance of such representations for spreadsheet interactivity and performance.

Source: Conversation with Bing, 08/05/2024 (1,4) Efficient and Compact Spreadsheet Formula Graphs <https://arxiv.org/pdf/2302.05482v1.pdf> (Tang et al., 2023). (2,4) Use of directed acyclic graphs (DAGs) to identify confounders in <https://academic.oup.com/ije/article/50/2/620/6012812> (Tennant et al., 2020). <https://doi.org/10.1093/ije/dyaa213>. (3) [2402.14853] NL2Formula: NL2Formula: Generating Spreadsheet Formulas from Natural Language Queries <https://arxiv.org/abs/2402.14853>.

5.7 Genealogy Trees

BING copilot Directed Acyclic Graphs (DAGs) are a type of graph that's particularly useful for representing structures that have dependencies, but where cycles cannot occur. This makes them ideal for analyzing genealogy trees, which trace family relationships in a way that naturally forms a DAG structure.

Here's how DAGs can be used in genealogy:

1. ****Representation**:** Each individual in a genealogy tree can be represented as a node in a DAG. The edges would represent the parent-child relationships, directed from parent to child.
2. ****Inheritance Patterns**:** By analyzing the paths in a DAG, one can study inheritance patterns across generations, such as the inheritance of genetic traits or family names.
3. ****Ancestor Analysis**:** DAGs can help identify common ancestors of individuals by finding the lowest common ancestor in the graph, which is the most recent individual from whom two or more individuals are descended.
4. ****Detecting Relationships**:** The structure of a DAG allows for the detection of various types of relationships, such as siblings, cousins, and more distant relations, based on the connectivity and paths between nodes.
5. ****Topological Sorting**:** This is a linear ordering of the nodes of a DAG, which can be used to order individuals chronologically or generationally.
6. ****Transitive Reduction**:** This process simplifies the DAG by removing all edges that are implied by transitivity without changing the reachability relation. In genealogy, this means showing the direct parent-child relationships without the intermediate ancestors.
7. ****Data Integrity**:** Since genealogy trees must not contain cycles (as an individual cannot be their own ancestor), DAGs can be used to validate the integrity of genealogical data.

DAGs are powerful tools in computational biology and data science, and their application in genealogy is just one of the many ways they can be utilized to model and analyze complex relationships and dependencies(2).

Source: Conversation with Bing, 08/05/2024 (1) Directed acyclic graph - Wikipedia. https://en.wikipedia.org/wiki/Directed_acyclic_graph. (2) An Introduction to Directed Acyclic Graphs (DAGs) for Data Scientists. <https://dagshub.com/blog/an-introduction-to-directed-acyclic-graphs-dags-for-data-scientists/>. (3) Analyze and Create Elegant Directed Acyclic Graphs - ggdag - GitHub Pages. <https://r-causal.github.io/ggdag/>. (4) undefined. <https://DAGsHub.com>.

5.8 Hierarchies including voltage graphs

Suppose we have an organisation with a strict chain of command. If the people in the organisation are the nodes then we can connect each person with a directed edge to their line manager. In principle, the boss would be the only sink node for this DAG and that might be absolutely true for a military or criminal organisation. Such a formal hierarchy would just be a tree with every node, except the sink node, having out-degree one so it is not a very interesting DAG.

Suppose now a social scientist is conducting interviews with employees and as part of that they ask each employee who they work with in practice and also where they rate their work

colleagues as higher or lower status. Now these links can cut across the tree of formal hierarchy as some employees may spend of of their time working with a range of different teams writhin teh company. It would even be possible for an employee to misjudge the position of others in the formal hierachy so this could potentially lead to a lack of a true DAG.

*** Voltages, weighted edges.

5.9 Data Compression

BING copilot Let's explore a data compression algorithm that leverages **Directed Acyclic Graphs (DAGs)** for efficient representation. DAGs find unconventional use in this context by compactly representing collections of sequences. Here's how it works:

1. **DAG Compression for Graphs:** - Researchers have introduced a simple data structure for storing graphs (and more general structures) in a compressed manner using DAGs(2). - Instead of directly encoding the edges of a graph, the DAG compression approach focuses on representing the graph's structure in a more compact form. - The key idea is to identify shared substructures (subgraphs) within the graph and represent them efficiently using a DAG.
2. **Linear Time Complexity:** - The beauty of DAG compression lies in its linear time complexity. Numerous standard graph problems can be solved in time linear to the size of the DAG compression of a graph, rather than the number of edges in the original graph(2). - This efficiency makes it an attractive approach for data compression, especially when dealing with large graphs.
3. **Optimal Top DAG Compression:** - Another interesting result involves constructing an optimal top DAG from an ordered node-labeled tree. The resulting DAG has a specific height and size, which is optimal and improves on previous bounds(3). - While this specific application focuses on trees, it demonstrates the power of DAGs in compressing structured data.

In summary, DAG compression provides an elegant way to represent graphs and other structures efficiently, making it a valuable tool for data compression and optimization tasks. (2,3).

Source: Conversation with Bing, 09/05/2024 (1) Faster Graph Algorithms Through DAG Compression. https://www.ifis.uni-luebeck.de/uploads/tx_wapublications/dag_compression_public.pdf. (2) [1712.05822] Optimal top dag compression - arXiv.org. <https://arxiv.org/abs/1712.05822>. (3) Practical Applications of Directed Acyclic Graphs - Baeldung. <https://www.baeldung.com/cs/dag-applications>.

5.10 Geometric embedding of data

(T)Geometric embedding of data. Effective Dimension of Data. (**end T**)

5.11 Sampling Data for a DAG

(T)Sampling, snowball. (**end T**)

5.12 Null Models and Data Randomisation

(T)Null models, configuration model. (end T)

Chapter 6

DAG Numerics

- DAG libraries
- Numerical Representation of a DAG
- Numerical Algorithms for DAGs
 - Topological sort
 - Configuration model for DAGs
 - Longest Path algorithm
 - Transitive Reduction
 - Antichain Algorithm
- Configuration model? Here or in models?
- Visualisation and Layout
- Removing cycles from directed graphs: Agony, other hierarchical methods.

There is usually no need to create your own numerical representation of a network in your code or to implement various standard network algorithms yourself. There are many well established libraries with all the functions you need. Their long history means they are well tested and there are many sources of documentation and help. At the time of writing **NetworkX** for **Python** or the **Boost** graph library for **C++** are good examples.

However there are times when the best known standard libraries are not the right solution. Code written in **Python** can be slow so a specialised library with fast code (perhaps based on **C** internally) might be a better option but this will tend to be less well supported and tested. The **Boost** graph library is excellent but it is complicated to learn. In some cases a problem may not need the sophistication and flexibility offered by established libraries. Sometimes, it is better to keep all the code needed within your project and not rely on external libraries, e.g. to ensure portability. In some rare cases, a simpler code might allow you to optimise for speed or memory though some libraries will offer excellent speed and memory performance that will be hard to beat.

Perhaps the best reason to learn a bit about the numerical implementation needed to investigate DAGs is that it offers the chance to understand how ‘black box’ libraries work in principle which will be invaluable when designing and debugging code based on a library. Learning how to implement the ideas from the previous chapters in numerical code is another good way to understand those ideas.

6.0.1 Pseudocode conventions

There is no one set of pseudocode conventions.

(T) Perhaps we can follow p.20 of Cormen et al. (2009). (end T)

For this chapter on numerical aspects of DAGs we will adopt a convention that our nodes are labelled by consecutive integers from 0 to $(N - 1)$. We will use indices i, j, \dots to indicate these integer node labels rather than u, v, \dots that we have used elsewhere to represent individual nodes.

In this chapter we will make use of array-like structures however these are not the structures, often called an array, of fixed number of dimensions encountered in the older languages and sometimes present in modern ones¹. Rather we will assume that we are using DYNAMIC ARRAYS whose size automatically changes when elements are added to the end of the array. For instance this kind of structure is produced by the `list` in Python, the `vector` found in the STL (standard template library) of C++, the `list` in Java (see “Introduction to Programming Using Java” section 10.2 Eck (2007)),

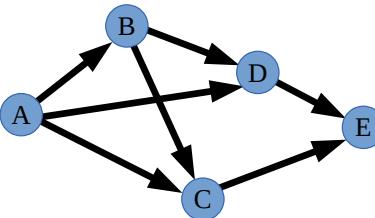
We will follow the notation of Python, so `array[i]` gives the i -th entry of the list with no limit on the length and i an integer² running from zero upwards. The value of `array[i]` could be a number or it could be a second list. For instance `array[i][n]` will indicate that we want the n -th index of the list `array[i]`.

6.1 The Numerical Representation of DAGs

6.1.1 Numerical array

We have already seen in section 1.1 and in Fig. 1.4 that we can define a network in terms of an adjacency matrix A_{ji} representing the weight of an edge from node labelled i to node labelled j . For DAGs, if we order the indices in a total order, (see section 2.6) then we get an lower-diagonal matrix which can have advantages for numerical matrix manipulations.

For example, the adjacency matrix A for the DAG in Fig. 2.1a is



$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \quad (6.1.1)$$

In this case we have linked the rows and columns correspond to the nodes in alphabetical order, i.e. first row lists the successors of node A , the second column gives the predecessors of node B and so on. As the labelling here has been chosen so that alphabetical order is one of two possible total orderings for the nodes of this DAG, we see our adjacency matrix is indeed lower triangular.

The adjacency matrix can be useful for analytical calculations in network science but for numerical work it is usually an inefficient way to represent a network as a matrix. Every missing edge is represented as an explicit zero stored so the storage requirements rise in proportion to the square of the number of nodes, $O(N^2)$. Worse, some operations, such as finding the

¹In fact in Python there is no array in the main language and these are only found as part of add-on packages such as the `numpy` package.

²There are also more sophisticated structures available that allow us to use any type of object as an index. For example, perhaps `i` is an instance of a `node` class that we have defined. These include the `dictionary` in Python and the `map` in Java and C++. We discuss appropriate structures for sets in computer languages in appendix A.4.2.

neighbours of a node, are very slow, proportional to the number of nodes in the network not to the average in- or out-degree. What this means is that unless the network is small, or it is large and dense, there are often real penalties in terms of time and memory if we are to work with the matrix representation of a DAG. These costs may be alleviated by using special sparse-matrix representations of matrices (i.e. missing edges are not explicitly stored) available in modern matrix packages and which will also use highly optimised algorithms to perform matrix multiplication, finding eigenvalues and so forth. So if the DAG is not too big it may be that the adjacency matrix approach can be useful numerically but it is not the most natural way to work with networks numerically. So we will now turn to a more natural representation of DAGs for most numerical work.

6.1.2 Adjacency List

The most efficient way to store a network in a computer or in a file is an ADJACENCY LIST. In this representation, we give a list of the nodes, and for each node we give the set of neighbouring nodes, even if it is an empty set. For the case of directed networks, such as DAGs, we can just list the successor nodes. It is possible in principle to deduce the predecessors when needed. In practice, that is slow so we would often repeat this with a list of the predecessors for each node. For example the network in (6.1.1) has an adjacency list of

source	successors	
A	B,C,D	
B	C,D	
C	E	
D	E	
E		

(6.1.2)

So in our code we would have a list of successors indexed by node, say `successorList`. Then each entry `successorList[i]` is a second list giving the successor nodes of the node with index i . So then `successorList[i][n]` is the n -th successor of node with index i . Every node is listed in `successorList`, even if it has no successors; `successorList[i]` is simply an empty list if the node with index i has no successors. Every edge is represented as one entry in one successor list of one node so there storage needs grow in proportion to the number of edges. Edges which don't exist are not explicitly represented so this approach uses the smallest amount of memory possible for a DAG. As we often need to access the predecessors of each node quickly, it is natural to have a second list of lists giving the predecessors, say `predecessorList[i][n]` is the n -th predecessor of node with index i . This will speed up many operations at the cost of doubling the memory requirement.

Overall the adjacency list good way to represent any network in a computer programme as it is sparse (only records edges which are present) and it is easy to find local properties of each node such as the degree. The downside is that the adjacency list is not ideal for algebraic manipulations.

Edge List Representation

An EDGE LIST is a simple list of the source and target nodes of each edge, with weight of edge if needed. For the network in (6.1.1) we would have

source	target	
A	B	
A	C	
A	D	
B	C	
B	D	
C	E	
D	E	

(6.1.3)

An edge list representation is good for storing a network as a data file since this is simple (often human readable) and it is sparse (only records the edges which are present). On the other hand an edge list is a poor way to store a network internally since finding the edges of one node and hence the neighbours would be a slow process. This way of storing a network also misses nodes with no edges though this is easily fixable with an extra data structure. Finally, this is not a good representation for algebraic manipulations.

6.2 Numerical Algorithms for DAGs

DAGs have been important in computer science for a long time so it is not surprising that some of the most important numerical algorithms needed to analyse DAGs are well established and widely known.

6.2.1 Topological Sort

(T) Call the algorithm a ‘topological sort’ but the order in a DAG a ‘topological order’? (end T)

There are two main ways to produce a total order in a DAG using what are called TOPOLOGICAL SORT algorithms. One way is to adapt a depth first search **DFS** (e.g. see Cormen et al., 2009, section 22.4) but here we will look at KAHN’S ALGORITHM (Kahn, 1962).

In practice, we need not physically remove the edge from the network, we just label it as ‘removed’ or ‘visited’. We visit each edge once in this algorithm as line 3 looks at every node U and line 5 looks at every edge E leaving that node. Thus the running time scales as the number of edges, $O(E)$.

The algorithm will produce a unique total order (often called a topological order in computing contexts) for the network if the node U chosen from the set Z in line 3 is the same every time the code is run. However, there are normally many distinct topological orders for any DAG and changing the way we select nodes U and V in lines 3 and 5 (e.g. at random) could produce different topological orders. One useful option is to use a different data structure for the set Z storing the nodes whose incoming edges have all been visited (removed in the language of the pseudocode Algorithm 1); QUEUE only connected to ??? of way do fix the tounique here e data structure used for the current set of nodes S in Algorithm 1 need not be a set.

```
indegree_map = {v: d for v, d in G.in_degree() if d > 0}
zero_indegree = [v for v, d in G.in_degree() if d == 0]
```

Algorithm 1 A Topological sort algorithm using Kahn's algorithm (Kahn, 1962). Adapted from algorithm presented on WIKIPEDIA'S PAGE ON TOPOLOGICAL SORT.

Require: Z = Set of all sinks (nodes with no incoming edge)

```

1:  $L$  = Empty list
2: while  $Z$  is not empty do
3:   remove a node  $U$  from  $Z$ 
4:   add  $U$  to  $L$ 
5:   for each node  $V$  with an edge  $E$  from  $U$  to  $V$  do
6:     remove edge  $E$  from the network
7:     if  $V$  has no other incoming edges then
8:       add  $V$  to  $Z$ 
9:     end if
10:    end for
11: end while
12: if network has edges then
13:   return error                                 $\triangleright$  network has at least one cycle
14: else
15:   return  $L$                                  $\triangleright$  a topologically sorted order
16: end if
```

```

while zero_indegree:
    this_generation = zero_indegree
    zero_indegree = []
    for node in this_generation:
        if node not in G:
            raise RuntimeError("Graph changed during iteration")
        for child in G.neighbors(node):
            try:
                indegree_map[child] -= len(G[node][child]) if multigraph else 1
            except KeyError as err:
                raise RuntimeError("Graph changed during iteration") from err
            if indegree_map[child] == 0:
                zero_indegree.append(child)
                del indegree_map[child]
    yield this_generation

    if indegree_map:
        raise nx.NetworkXUnfeasible(
            "Graph contains a cycle or graph changed during iteration"
        )
```

Alternatively, and this is how we will do the TR we use DFS.

Depth-first search (Wikipedia) An alternative algorithm for topological sorting is based on depth-first search. The algorithm loops through each node of the graph, in an arbitrary order, initiating a depth-first search that terminates when it hits any node that has already been visited since the beginning of the topological sort or the node has no outgoing edges (i.e., a leaf node):

The pseudocode for DFS from Wikipedia is

```

L <- ' Empty list that will contain the sorted nodes
while exists nodes without a permanent mark do
    select an unmarked node n
    visit(n)

function visit(node n)
    if n has a permanent mark then
        return
    if n has a temporary mark then
        stop  (graph has at least one cycle)

    mark n with a temporary mark

    for each node m with an edge from n to m do
        visit(m)

    mark n with a permanent mark
    add n to head of L

```

6.2.2 Longest Path algorithm

(T)Other paths (**end T**)

(T)Height, depth (**end T**)

(T)Intervals (**end T**)

6.2.3 Transitive Reduction algorithm

Aho, Garey & Ullman (1972)

TR (Wikipedia) When measured both in terms of the number n of vertices and the number m of edges in a directed acyclic graph, transitive reductions can also be found in time $O(nm)$, a bound that may be faster than the matrix multiplication methods for sparse graphs. To do so, apply a linear time longest path algorithm in the given directed acyclic graph, for each possible choice of starting vertex. From the computed longest paths, keep only those of length one (single edge); in other words, keep those edges (u,v) for which there exists no other path from u to v . This $O(nm)$ time bound matches the complexity of constructing transitive closures by using depth-first search or breadth first search to find the vertices reachable from every choice of starting vertex, so again with these assumptions transitive closures and transitive reductions can be found in the same amount of time.

This is how NetworkX does this.

```

def dfs_edges(self, source):
    visited = set()
    visited.add(source)
    stack = [(source, source.out_neighbour_list)]
    depth_now = 1
    while stack:
        parent, children = stack[-1]
        for child in children:

```

```

        if child not in visited:
            yield parent, child # this generates the edge that is returned.
            visited.add(child)
            stack.append((child, child.out_neighbour_list))
            depth_now += 1
            break
    else:
        stack.pop()
        depth_now -= 1

def calc_transitive_reduction(self):
    descendants = {}
    # count before removing set stored in descendants
    check_count = {}
    for paper in self.paper_list:
        check_count[paper] = paper.get_in_degree()
    for u in self.paper_list:
        u_nbrs = set(u.out_neighbour_list)
        for v in u.out_neighbour_list:
            if v in u_nbrs:
                if v not in descendants:
                    descendants[v] = {y for x, y in self.dfs_edges(v)}
                u_nbrs -= descendants[v]
        check_count[v] -= 1
        if check_count[v] == 0:
            del descendants[v]

        u.tr_out_neighbour_list = list(u_nbrs)
        u.tr_in_neighbour_list = []
    for u in self.paper_list:
        for v in u.tr_out_neighbour_list:
            v.tr_in_neighbour_list.append(u)

```

6.2.4 Antichain Algorithm

6.3 Randomising DAGs

We discussed in section 1.5 how random network models are often used to randomise a network subject to some constraint. This so the randomised network can act as a null model giving a benchmark for any measurements made on the original network. In particular, we noted in section 1.5.2 that the configuration model allows us to create a network with any given degree distribution but where the edges are otherwise scattered at random. One popular way to obtain networks of this type is to swap pairs of edges. For a directed network, we have to make sure that we swap the same end of two edges (say both ends are arriving at a node) but otherwise this edge swapping works as before. For directed graphs this now preserves both the in- and out-degree of each node.

In principle then we can use the same trick, swapping the target nodes of two edges so (u, v) and (w, x) are removed and are replaced by edges (u, x) and (w, v) . However, the acyclic property of a DAG lies at the heart of what it is encoding in the data. So invariably, we want to preserve the acyclic nature of the network when randomising a DAG, say to provide a null

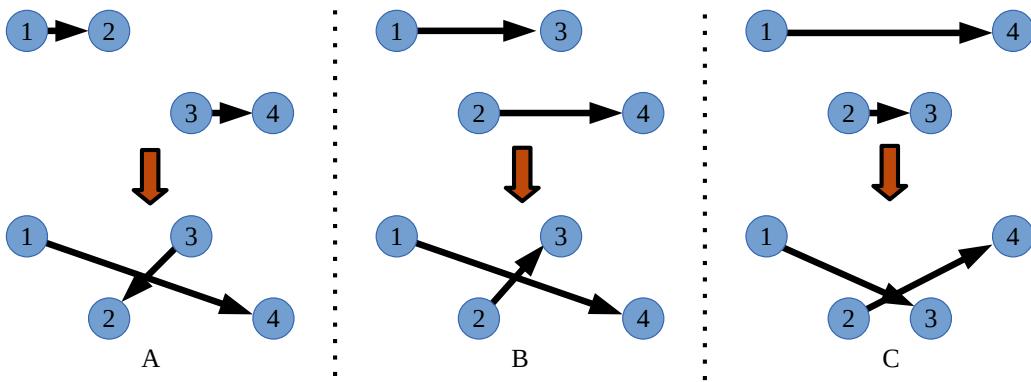


Figure 6.1: Three scenarios seen when swapping edges in a DAG. We have a total order for the DAG. In the figure the numbers on the nodes indicate the relative position of each node in the total order such that edges only run from low numbered nodes to higher numbered nodes. For any two chosen edges, the four nodes in these edges may appear in the total order in just three distinct ways.

The cases where edge swaps are of the form shown in the middle and right diagrams cause no problems. Since the new edges respect the total order, these examples cannot generate a cycle and so the network remains a DAG after these rewirings. Note that other total orders may exist and can be destroyed by these rewirings.

The rewiring case on the left, however, does violate the total order. This may or may not create a cycle. If we can find another total order where the labelling of these four nodes is of the type in the middle or right, then again no cycles are produced. We have, however, destroyed this (and possibly other) total order. On the other hand, if our chosen total order plays a definitive role, e.g. it is the publication date of documents in a citation network, then we may wish to exclude the left-hand case when rewiring a DAG.

models. Preserving acyclicity provides us with an extra constraint on top of keeping the degree of each node unchanged.

The simplest way to ensure we preserve the acyclic nature of a DAG under edge swapping (for example see Karrer and Newman (2009), Carstens (2014)) is to first find a total order. Once we have that, there are three cases we need to consider, as shown in Fig. 6.1. In two cases, we are guaranteed that rewiring a pair of edges will maintain the acyclic properties because it guarantees that the chosen total order is still a total order for the rewired network. However the third type of edge swap, case A in Fig. 6.1, is more problematic. If we remove edge (u, v) and (w, x) and replace them with edges (u, x) and (w, v) , then the second of these new edges is no longer in alphabetic order. If the alphabetic order of the node labels reflects the total order we choose, then the second new edge is now violating this total order since w comes after v . This means the acyclic nature of the resulting network is not guaranteed by the original total order. However, sometimes this edge swap does not create a cycle and the resulting network is still a DAG. One way to see this is that there could be another total order where w is still before node v . An example of this is shown in Fig. 6.2. Equally, in some cases, this problematic edge swap does give a network with a cycle. To find out requires a lot more work.

So if it is important to also maintain one particular total order, we just choose not to make edge swaps of type A in Fig. 6.1, we just work with edge swaps of type B and C in Fig. 6.1. We could do this even if the total order chosen had no significance but then the rewired DAGs would give prominence to this total order and this may well be inappropriate for context. This approach has a bias in the networks produced by rewiring limiting their use as a null model. Another approach to this issue could be to impose more constraints. One such example follows in the next section.

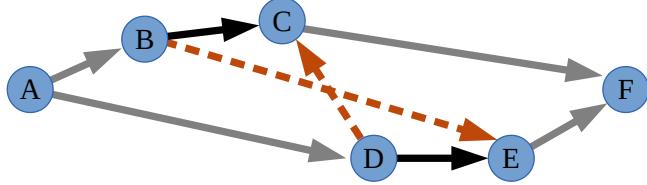


Figure 6.2: An example of an edge swap in the DAG of type A in Fig. 6.1. The alphabetical order of the labels of the nodes gives one of six different total orders for this DAG where edges run from nodes with labels earlier in the alphabet to nodes with labels later in the alphabet. The rewiring of (B, C) and (D, E) to become (B, E) and (D, C) is rewiring breaks that total order but in this case still leaves us with a DAG. The reason is there are four other total orders which would be preserved. For instance in another total order we can label the top edge as going from B to D so the bottom edge is from C to E . Now this rewiring is of type B from Fig. 6.1. Only if the original order in the labels is important to preserve, e.g. publication data of documents in a citation network, do we need to exclude this rewiring.

6.3.1 Level Preserving DAG Rewiring

(TODO) Rewrite. From `ChineseCharactersMethods.tex`, part of the Chinese Characters project on Overleaf. Randomising a DAG subject that also preserves height. (end TODO)

Apart from maintaining the acyclic nature of our network, another key property we might wish to maintain is the height $h(v)$ of each node v (see section 2.5). However, a rewiring such as the safe edge swapping of types B and C in Fig. 6.1, can leave us with a DAG yet change the heights of some of the nodes. For instance in Fig. 6.3 if we replace edges (E, I) and (C, F) with edges (E, F) and (C, I) we would have a DAG but the height of node F changes from one to two because of the path (B, E, F) . So what we are looking for is criteria which will maintain acyclicity and the heights when we perform an edge swap.

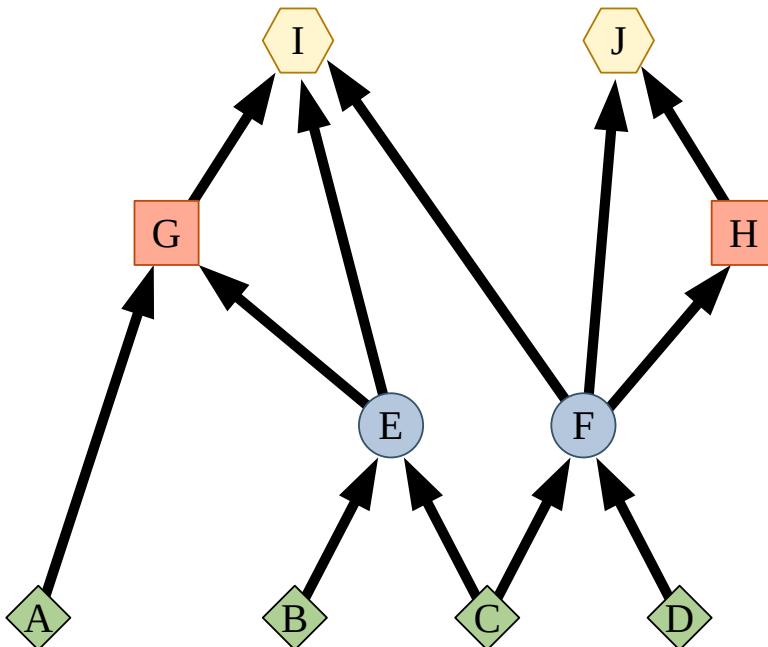


Figure 6.3: Example of graph structure. Edges (E, I) and (F, J) are removed under transitive reduction. So $\{A, B, C, D\}$ (green diamonds) are sources of height zero, the $\{E, F\}$ (blue circles) are height 1, the height two nodes are $\{G, H\}$ (red squares), while I and J (yellow hexagons) are the height 3 sink nodes. If we rewire by replacing edges (E, I) and (C, F) with the edges (E, F) and (C, I) , we would still have a DAG but the height of node F changes from one to two.

To explain how to rewire an edge pair without changing heights, we will consider the edge swap where edges (u, x) and (v, y) are removed but they are replaced by edges (u, y) and (v, x) . We will assume that these new edges do not already exist so we do not allow multiedges.

The first point to make is that the heights of the source nodes in the rewiring, u and v , remain unchanged. The longest paths from a source node of the DAG to u and v will never contain any edge leaving u and v and so these will be unchanged. This means the height of u and v are unchanged. The issue is the height of the target nodes in this swap, nodes x and y .

So the first test we will do is a necessary condition to preserve height and that is we can only rewire an edge pair (u, x) and (v, y) if $h(u) < h(y)$ and $h(v) < h(x)$. The reason for this is that the height of node y after rewiring will be $h(y) \geq h(u) + 1$. This is because the height of u is unchanged so there is at least one path from a DAG source node to y via u that has length $h(u) + 1$: the longest path to u that defines its height plus the new edge from u to y . Since we do not want the heights to change after rewiring, the current height of y must be equal in length or longer than the length of this new path we will create. So we know that $h(y) \geq h(u) + 1$ and thus we can reject a potential edge swap if $h(u) < h(y)$. A similar argument for nodes v and x gives the requirement that $h(v) < h(x)$.

Note this excludes some rewirings allowed by assigning a total order as we can have $h(u) = h(y)$ and/or $h(v) = h(x)$ yet $t_u < t_y$ and $t_v < t_x$. (T)³ For example in Figure ?? rewiring (E, I) and (C, F) to (E, F) and (C, I) leaves us with a legal DAG but the height of node F changes. This condition alone will not preserve height however but it is a necessary condition.

Note that this condition will always exclude case *A* in Fig. 6.1 from the possible rewirings so this also guarantees that any accepted this edge swap will give us a new DAG.

The second test is to note that if the existing edges (u, y) and (v, x) are both between nodes of heights differing by two or more, then this edge pair can be rewired. Such edges can be removed as they are not part of any longest path so the heights of any node do not change when these edges are removed. So when we add the two new edges, heights can only stay the same or increase. When we add the new edge (u, y) there is a new path from a DAG source node to y which takes one of the longest paths to u and then adds the new edge to give a new path to y . So this route can only lead to the new height of y being one more than u . However our first test ensures the height of y was at least the height of u plus one. So this new potential path cannot increase the height of node y , it can only at best match it. So edge swaps obeying tests 1 and 2 do not change heights.

The third test is that we can always rewire if $h(x) - h(u) = 1$ and $h(y) - h(v) = 1$. To understand this we start by noting that the first condition combined with this one ensures that $h(u) = h(v)$. The new edges give new paths to y such as the longest path to u then the new edge (u, y) . Such a path has length $h(u) + 1$ but we see in this case this is just $h(v) + 1 = h(y)$, the original height of y . So even if the edge swap disrupts the longest path to y that set the height originally (so the height would drop after edge removal), the new paths formed by swapping give y the same height it had before. So there can be no change in the height of y and, by the same arguments, the height of x .

For instance consider (E, G) and (F, H) in Fig. 6.3. Both are the only edges linking height 2 nodes G and H to the height below as defined by the paths $\{B/C, E, G\}$ and $\{C/D, F, H\}$. However after the swap the heights are now defined by $\{B/C, E, H\}$ and $\{C/D, F, G\}$ so the heights are preserved.

The fourth test is as follows. If $h(x) - h(u) = 1$ but $h(y) - h(v) > 1$ then we can only rewire if and only if there is a different predecessor of x (so not u), say w , which has height one less than x . That is we require $h(w) = h(u)$ and $(w, x) \in \mathcal{E}$ in this case.

In this fourth test, the existence of this predecessor w ensures that there is still a longest path of length $h(x)$ to x even after the edges are removed. Adding in the edge (v, x) can at

³(T)Fix the notation for a total order.

most give a new path of length $h(v) + 1$ from a global source. However we know from test one that $h(v) < h(x)$ so this new path can match but can never be longer than the path to a DAG source node to x via w . So the height of x does not change. For the height of y we can use the argument of the third test to show it is unchanged.

An example of this would be (C, F) and (A, G) in Fig. 6.3. We can swap (C, F) as the edge (D, F) preserves the height of F .

Finally, the fifth test is that if $h(x) - h(u) > 1$ but $h(y) - h(v) = 1$ then we can only rewire if and only if there is a predecessor of y , say w , which is also just one height below so $h(w) = h(v)$ and $(w, y) \in \mathcal{E}$. This is just the fourth test with the labels of the nodes exchanged.

(T)⁴

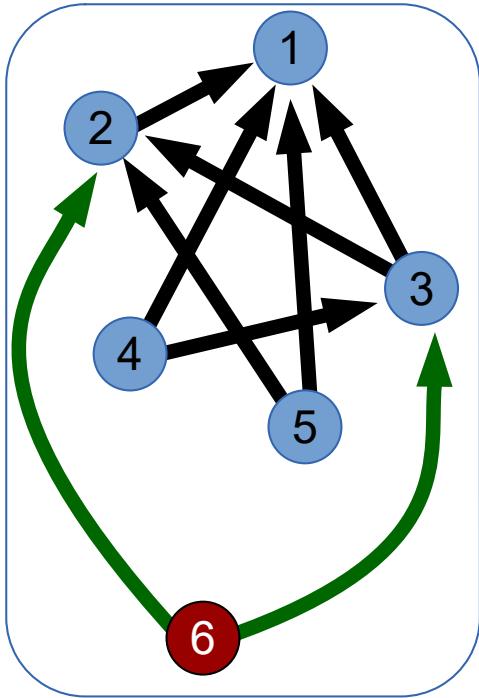
So now we see we have covered all the possibilities and, if we exclude multiedge DAGs, then we have five tests we can apply in order to establish if a proposed edge swap preserves both height and the in- and out-degree of the DAG. To summarise, if we consider replacing edges (u, x) and (v, y) with (u, y) and (v, x) and all these nodes are distinct (otherwise nothing happens in the rewiring) then we follow the following algorithm in the order set out below:-

0. If edges (u, y) and (v, x) already exist we do not rewire.
1. If $h(u) \geq h(y)$ or $h(v) \geq h(x)$ we do not rewire.
2. If none of the current edges span exactly one height, so $h(x) - h(u) > 1$ and $h(y) - h(v) > 1$, then we can rewire.
3. If $h(x) - h(u) = 1$ and $h(y) - h(v) = 1$ then we can swap these two without further checks.
4. If $h(x) - h(u) = 1$ but $h(y) - h(v) > 1$ then we can only rewire if and only if there is a predecessor of x , say w , which is also just one height below so $h(w) = h(u)$ and $(w, x) \in \mathcal{E}$.
5. If $h(x) - h(u) > 1$ but $h(y) - h(v) = 1$ then we can only rewire if and only if there is a predecessor of y , say w , which is also just one height below so $h(w) = h(v)$ and $(w, y) \in \mathcal{E}$.

6.4 Numerical Implementation of Price model

There is a very simple but effective trick one can use to implement the Price model. The main problem comes in choosing target nodes from the existing graph with a probability Π of the form (4.8.3) that is proportional to degree. One simple way to do this is to keep an ATTACHMENT LIST which contains each node i a total of $N\Pi(k_i^{(\text{in})}, t) = (k_i^{(\text{in})} + \mu)$ times. That way when we choose uniformly at random from this list we will be choosing any one node i with probability $\Pi(k_i^{(\text{in})}, t)$. The simple way to fill this list correctly is to first add each new node μ times to this attachment list. Then every time a node i is chosen as the target node for a new edge, this target node i is added once to the attachment list. One limitation of this scheme is that μ must be an integer whereas in principle this parameter can be any positive real number. This scheme also has a penalty in terms of memory. However, the speed and simplicity means this attachment list approach is often an effective way to implement the Price model. This algorithm also allows several more variations of the model to be implemented with relative ease, e.g. having a variable fitness μ_i for each node.

⁴(T)What is essential to maintain the heights of nodes is that there is always *one* edge linking a node u to one predecessor r which has height exactly one less than u (i.e. the edge length of (r, u) is one). More formally $\exists r$ s.t. $h(u) = h(r) + 1$. If there is only one edge of this type and we are considering it for rewiring then we have to be careful though we may still be able to rewire such an edge. So we have a new set of edges \mathcal{E}_1 which span nodes separated by one height and for each node we have a set of predecessor nodes (past neighbourhood) which are only one height below, say $\mathcal{N}_1^-(v) = \{u | (u, v) \in \mathcal{E}, h(v) = h(u) + 1\}$.



$$L(5) = [1, 2, 1, 3, 1, 2, 4, 1, 3, 5, 2, 1]$$

$$L(6) = L(5) \cup [6, 3, 2]$$

Figure 6.4: Illustration of how the attachment list algorithm for the Price model works for a case with $m = 2$ and $\mu = 1$. The initial directed acyclic network, $\mathcal{D}(t = 5)$, is shown with blue circles for the nodes and labelled with time at which they were created. The black lines are the edges in $\mathcal{D}(t = 5)$. We show one new node, red number 6, being added along with two new edges, green lines, so $m = 2$ here. The process for adding the new edges in this algorithm is as follows. The attachment list starts with $L(5)$ shown in the figure on the right. Choosing twice uniformly at random from this list $L(5)$, in this instance we are given nodes 2 and 3 as the targets for edges from new node 6. The node 6 is then added once to the attachment list as $\mu = 1$ and nodes 2 and 3 are each added once to the attachment list to reflect the fact that their indegree has increased by one. So the sequence [6, 3, 2] is used to extend $L(5)$ to produce a new attachment list $L(6)$ to be used when choosing the edges to add to node 7.

6.5 Visualisation

- TR/HASSE DIAGRAM. Essential edges (Links) and non-essential edges.
- Using the height and depth of a node as coordinates for a node are a convenient way to start visualising a DAG.
- Anything in (Zhao and Strotmann, 2015)?
- From Ugwu and Cuffe (2023): “layered graph drawing [5], [6] of the corresponding DAG. These graph drawing algorithms position each node within a two-dimensional canvas such that all oriented edges are aligned, and therefore all power flows are from the top towards the bottom of the diagram. Rather than depicting the geographical positions of the grid’s buses, such a diagram explicitly shows the bulk power transactions arising from the prevailing generator dispatch and system loading [5] P. Healy and N. S. Nikolov, How to Layer a Directed Acyclic Graph, M. P., J. M., and S. Leipert, Eds. Springer, 2001, vol. 2265, pp. 16–30. [6] E. Damiani, B. Oliboni, E. Quintarelli, and L. Tanca, “A graph-based meta-model for heterogeneous data management,” Knowledge and Information Systems, vol. 61, pp. 107–136, 1 Oct. 2019, issn: 02193116. doi: 10.1007/s10115-018-1305-8.”
- Package review (Pitts and Fowler, 2024):- TikZ, DAGitty, ggdag, dagR, and igraph

The nodes do not in general have any coordinates in space. When we make a two-dimensional representation of a network in a figure, we may place the nodes anywhere provided we always connect nodes as specified by the edges of the network. Visualisation of networks is both an art and a science. The presentation of any data can make or break a piece of work. While pictures can help or simply be aesthetically pleasing, there is a danger, especially with large networks, to produce what are sometimes called “SPAGHETTI MONSTERS”, figures that do not convey any useful information.

Visualisation of networks is a whole subfield in its own right. Displaying the information in a graph of a few tens of nodes is already challenging, working with hundreds or thousands often results in meaningless monsters unless a great deal of effort is made.

The order inherent in DAGs leads to some natural features.

Using the and of a node as coordinates for a node are a convenient way to start visualising a DAG, see Fig. 6.5.

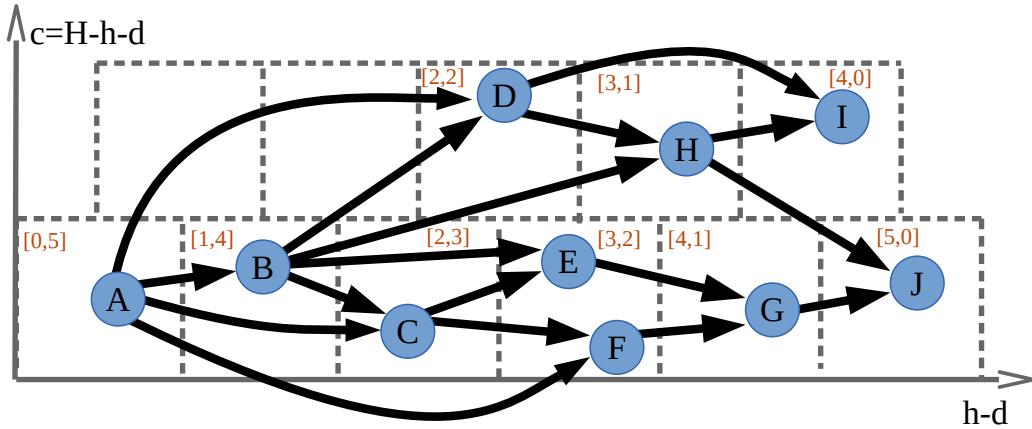


Figure 6.5: Using height depth height and depth as coordinates when visualising for a DAG. In this case the horizontal axis is height minus depth ($h-d$) while the vertical axis is criticality $c = H-h-d$ of (3.2.1). Each square cell indicated by grey dashed lines is centred at coordinates for a different height h and depth d , shown in red as $[h, d]$ in each cell. nodes are placed in the square cell corresponding to the height and depth of that node. With the cell, nodes are moved to aid clarity. This is the same DAG as Fig. 2.6.

(T)Hasse diagram as poset visualisation (end T).

6.6 Removing Cycles

For any data sets with a natural order and a network representation, the DAG is an obvious way to capture that data and its order. However, in many cases the first network we create will be directed but will contain some cycles. This could be because of errors in the data. Perhaps in processing the logic in formulae in a large spreadsheet or a set of dependencies in software, the translation into a network was imperfect even though the original data was indeed a DAG. Perhaps the order in the system was strong but imperfect. For instance documents are rarely produced at a single moment in time; they go through many drafts and can be published in different formats at different times. So citation network often contains cycles.

One answer to these problems is to keep the cycles in place and then we must turn away from DAGs to other approaches to analyse these networks. Many of the properties we find in DAGs have analogues in directed systems. In particular, there is a lot of discussion of finding hierarchy in networks, both directed and undirected. However, what these methods are doing is imposing a DAG structure on these non-DAG networks. A hierarchy implies that there is a natural partial order. A partial order imposed on a graph defines a DAG. So another way to look at such methods is that they are finding ways to eliminate the cycles from a graph and to impose a sense of direction and order so that we have in effect a DAG. So these methods open up a second approach to analysing networks with cycles which is to eliminate the cycles by such methods and then to analyse the data as a perfect DAG.

6.6.1 Node Value Methods

The simplest way to remove cycles is to use some real valued data that we have on the nodes in order to define a total order. Suppose we have some real value $M(v)$ for every node $v \in \mathcal{V}$ of a general network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. This could be metadata, extra information on the network that is not encoded in the network topology, such as the publication date of a document in a citation network, the size of an animal in a foodweb, or the number of hits on a webpage. The node value $M(v)$ could also come from something we calculate from the network itself, such as the degree of each node or other centrality value (for example this is part of the method used by Shi et al. (2024)).

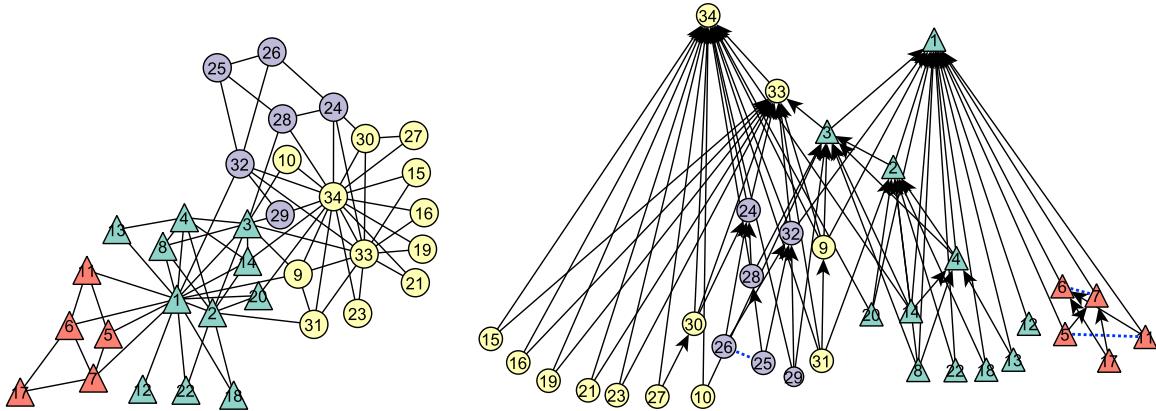


Figure 6.6: On the left is the Zachary karate club network (Zachary, 1977). The nodes are people involved in the club and the undirected edges represent binary interactions between these people. The node labels are the index used by Zachary (1977) while the shapes indicate the two communities found by Zachary (1977) using the Ford-Fulkerson binary community algorithm. The node colours indicate the four communities which give maximum modularity (Agarwal and Kempe, 2008). On the right is the DAG found by using the degree of the nodes in the network on the left to produce a partial order for the nodes on the right. The vertical height of each node is roughly related to degree so edges always point up since the edges are from lower degree to higher degree node. There are three edges where the nodes have equal degree, (6, 7), (5, 11) and (25, 26), and these have been dropped in the DAG on the right. Alternatively we could find another way to break the tie in order to give a total order and add a direction to these three edges e.g. using another node centrality measure such as betweenness. Note how node 1 has a lower degree than node 34, so $1 \leq 34$, but this does not ensure there is a path between the two in the DAG and we find there is none, so $1 \not\leq 34$.

We then create a DAG $\mathcal{D} = (\mathcal{V}_D, \mathcal{E}_D)$ with the same set of nodes $\mathcal{V}_D = \mathcal{V}$ but where every edge $(u, v) \in \mathcal{E}_D$ respects the order from the metadata values. That is if an edge $(u, v) \in \mathcal{E}$ in the original network \mathcal{G} , then edge exists in the DAG and is directed from u to v if $M(u) < M(v)$. More formally, if there are no ties then our $M(v)$ values have defined a total order on our network. Fig. 6.6 illustrates this idea using the degree of each node in the karate club network of Zachary (1977).

If there are ties, $M(u) = M(v)$ for distinct nodes u and v , we have defined a partial order \leq on our node set: $u \leq v$ if $M(u) < M(v)$ or, so that we have reflexivity, $u = v$. This is not always the same as our usual DAG partial order where $u \preceq v$ if there is a path in the DAG from u to v as there is no guarantee that there is a path from a higher degree node to a lower degree node. Nodes 1 and 34 in Fig. 6.6 illustrate this case as node 1 has a lower degree than node 34 yet there is no path linking these nodes in the DAG.

In some cases, we may have ties in our node values, $M(u) = M(v)$ for distinct nodes u and v . This will be most common when we have a very limited range of node values $M(u)$, such

as when this the node values are integers as in Fig. 6.6. Using the degree of each node as our $M(u)$ value is a simple example of this. The simplest way to deal with this is to drop any edge (u, v) when $M(u) = M(v)$. This means the number of edges in the DAG may be less than that in the original network. Alternatively, we could use a secondary condition, say another node centrality measure, to break ties in a consistent manner and ensure there are no cycles.

In many examples the original graph will be directed. Perhaps it ought to be a true DAG but errors in the data means there are some cycles present, e.g. a citation network. Then the interesting question is what to do with directed edges from u to v in the original network \mathcal{G} where the metadata points for a minority of edges points in the ‘wrong’ direction so that $M(u) > M(v)$? There are two basic options: drop these bad edges or flip the direction of the edge. Formally, if we drop edges then we define set of edges \mathcal{E}_D in the DAG so that

$$\mathcal{E}_D = \{(u, v) | (u, v) \in \mathcal{E}, M(u) < M(v)\} \quad (6.6.1)$$

If we flip the direction then

$$\mathcal{E}_D = \{(u, v) | (u, v) \in \mathcal{E}, M(u) < M(v)\} \cup \{(v, u) | (u, v) \in \mathcal{E}, M(u) > M(v)\} \quad (6.6.2)$$

Take the example of a citation network. We might have a collection of articles focussing on a common theme. During the editorial process, the editors might encourage references between the distinct papers in the collection to make the total collection more than the sum of its parts. So when the collection is published, all the articles in the collection will have the same publication date yet there could be several citations between the different articles. In fact, when we look at a real citation network, such ‘acausal’ edges are rare. Clough and Evans (2014) report that less than 1% of the edges in their citation networks are acausal.

There is no need to be so absolute here. We could use a stochastic process to choose between the drop or flip options. For instance we could accept an edge $(u, v) \in \mathcal{E}$ with probability $2[\exp(\beta(M(u) - M(v))) + 1]^{-1}$ for some fixed value of the parameter β as long as we flip the direction of the edge when one with $M(u) > M(v)$ is accepted.

Since every edge $(u, v) \in \mathcal{E}_D$ in the DAG satisfies $M(u) < M(v)$ the inherent order in the metadata values guarantees we will have no cycles in the graph \mathcal{D} .

The ties in the metadata, where $M(u) = M(v)$, might be a problem if these are too frequent. The simplest way to solve this is to add small, possibly random, values to the metadata, to give new values $M'(u)$ which has no ties.

6.6.2 Finding Implicit Hierarchy

(T)More extensive notes are in `hierarchy.tex` in the `Research/Hierarchy` directory. **(end T)**

In practice most citation graphs are not exactly DAGs with certain edges the ‘wrong’ way. In general there is a classic computer science problem looking at how to convert a directed graph to a DAG, i.e. how to remove edges to remove all the cycles. One can look at this in different ways. For instance the related problems are the Feedback Arc Set (FAS) problem which is identical to (may be) the Maximum Acyclic problem. However in general any rating/ranking rank aggregation method is doing this. If we imagine that we have pairwise comparisons between some or each of a set, this defines a directed graph (say from better to worse object/team) but we usually want an absolute ranking, i.e. a linear (total) order which in turn defines a DAG. As an intermediate step we can imagine removing edges and/or assigning new directions to edges until a DAG is produced and then defining a linear order form that somehow. Other approaches to rank aggregation include the use of AGONY Gupte et al. (2011), Tatti (2015, 2017), Letizia et al. (2018), a spring based approach Bacco et al. (2018).

(T)In Zakhlebin and Horvat (2017) (NOTE unpublished) a variety of methods are used. The Borda count (B_{ir} is the number of nodes with a larger rating r than node i , followed by a sum across all ratings to produce a final score). The Copeland matrix says node i dominates j , denoted $M_{ij} = 1$, if i is ranked higher by the majority of ratings. The Copeland's method computes the scores the difference between in- and out- degree $C_i = k_i^{(\text{out})} - k_i^{(\text{in})} = \sum_j (M_{ij} - M_{ji})$. Finally there is the Insertion sort (IS) algorithm. **(end T)**

SpringRank

(T)Based on `hierarchy.tex`. DO we use i, j or u, v as indices? **(end T)**

The SPRINGRANK method (Bacco et al., 2018) assigns a rating r_v to nodes v in a directed network by minimising the total energy of the following Hamiltonian⁵ H

$$H(\mathbf{r}) = \frac{1}{2} \sum_{i,j} A_{ij} (r_i - r_j - R)^2. \quad (6.6.3)$$

The picture is that r_i represent the coordinates of node i in a one-dimensional real space, $r_i \in \mathbb{R}$. Every directed edge (j, i) from node j to node i is associated with a perfect spring with spring constant A_{ij} and where the force on node i is $A_{ij}(r_i - r_j - R)$ in the negative \mathbf{r} direction while node j feels an equal and opposite force. Note that this means that the length of each spring when it is not in compression or tension is fixed to be R for all springs. The distance R sets the length scale but it can be factored out so without loss of generality we will set it to be $R = 1.0$ for simplicity. So if nodes can be positioned such that all springs have no tension or compression, $r_i - r_j = R = 1.0$, then the Hamiltonian achieves its minimum value of zero.

Interestingly, we can rewrite the solution for the minimum energy \mathbf{r} in terms of a *symmetrised* network, in which we add an edge in the reverse direction for every edge in the original network. This symmetrised network has the adjacency matrix \mathbf{S} where

$$\mathbf{S} = \mathbf{A} + \mathbf{A}^T. \quad (6.6.4)$$

This symmetrised adjacency matrix \mathbf{S} is then used to construct a standard matrix known as the graph LAPLACIAN \mathbf{L} (for example see Coscia (2021))

$$\mathbf{L} = \mathbf{K} - \mathbf{S}, \quad (6.6.5)$$

$$K_{ij} = (k_i^{(\text{in})} + k_i^{(\text{out})})\delta_{ij}, \quad (6.6.6)$$

where

$$k_i^{(\text{in})} = \sum_j A_{ij}, \quad (6.6.7)$$

$$k_i^{(\text{out})} = \sum_j A_{ji}. \quad (6.6.8)$$

The positions of the nodes r_i that form the lowest energy configuration can be solved by solving the linear equation

$$\mathbf{L}\mathbf{r} = \mathbf{J} \quad (6.6.9)$$

⁵Note that we are not looking at a dynamical process for which we would have to add a term summing over the kinetic energy contribution $\frac{1}{2} \sum_i \dot{r}_i$. Given we introduce a time coordinate below, it is important to note the time in these \dot{r}_i derivatives is not the same time as the time coordinates we use below.

where \mathbf{L} is the same Laplacian for a symmetrised adjacency matrix as defined in (6.6.5) and the external *source vector* \mathbf{J} is the vector of the difference out- and in-degree for each node (degree imbalance)

$$J_i = k_i^{(\text{out})} - k_i^{(\text{in})}. \quad (6.6.10)$$

Note that we always have

$$\sum_i J_i = 0. \quad (6.6.11)$$

Also we see that apart from the vector \mathbf{J} , the asymmetry, the directionality, of the original network is not in the Laplacian \mathbf{L} . Only the external source vector \mathbf{J} in (6.6.9) has any memory of the directionality. Working with this Laplacian of a symmetric is a familiar problem in network science so there is a lot of information about this equation.

The graph Laplacian matrix \mathbf{L} has a number of well known properties. For simplicity we assume that the network is weakly connected, that is the network represented by the symmetric $\mathbf{S} = \mathbf{A} + \mathbf{A}^T$ matrix has a single component. As \mathbf{L} is symmetric the left- and right-eigenvectors may be chosen to be identical and orthogonal. From the definition of $k_i^{(\text{in})}$ and $k_i^{(\text{out})}$ we can see that $\sum_j M_{ij} = 0$ which tells us that one eigenvector where all entries are identical has an eigenvalue of zero. We also know that this is the eigenvalue with the lowest real part so all other eigenvalues have larger positive real parts. Because of this eigenvalue we cannot invert \mathbf{L} . This also means that the rating vector \mathbf{r} is invariant under shifts by a constant, that is if r_j is a solution then so is $r_j + c$ for any constant c . This is because $\sum_j L_{ij}c = 0$ as noted above.

So we can write our solution in terms of the eigenvectors $\mathbf{e}^{(n)}$ and eigenvalues λ_n of the Laplacian \mathbf{L} . Let $n = 0$ represent the zero eigenvalue $\lambda_0 = 0$ so $e_i^{(0)} = 1$ and the other eigenvalues from $n = 1$ to N are larger with positive real parts (ordered so that $\lambda_n < \lambda_m$ if $n < m$) and these eigenvectors all satisfy $\sum_i e_i^{(n)} = 0$ (for $n \neq 0$, orthogonality with $\mathbf{e}^{(0)}$). We decompose the solutions and the external sources

$$\mathbf{r} = \sum_{n=1}^{N-1} \rho_n \mathbf{e}^{(n)}, \quad \mathbf{J} = \sum_{n=1}^{N-1} j_n \mathbf{e}^{(n)}. \quad (6.6.12)$$

We know that the $n = 0$ term does not contribute to the source term and hence deduce that ρ_0 is an arbitrary constant. Hence we can see that

$$\rho_n = \frac{j_n}{\lambda_n} \quad \text{for } n > 0. \quad (6.6.13)$$

and the solution may be written as

$$\mathbf{r} = \rho_0 + \sum_{n=1}^{N-1} \frac{j_n}{\lambda_n} \mathbf{e}^{(n)}. \quad (6.6.14)$$

So in general the result is dominated by the eigenvector of the second smallest, the smallest non-zero, eigenvalue λ_1 .

Formally we can write out the solution to (6.6.9) in terms of the PSEUDO-INVERSE $\tilde{\mathbf{L}}$ (the MOORE-PENROSE INVERSE)

$$\mathbf{r} = \tilde{\mathbf{L}}^{-1} \mathbf{J}. \quad (6.6.15)$$

The easiest way to define this pseudo-inverse $\tilde{\mathbf{L}}$ is to use the diagonalisation of the Laplacian matrix \mathbf{L} where, in most cases, we can write $\mathbf{L} = \mathbf{V}\Sigma\mathbf{V}^{-1}$. The matrix \mathbf{V} is a matrix whose columns are the left-eigenvectors while \mathbf{V}^{-1} is related to the right-eigenvectors. The Σ is a diagonal matrix with entries equal to the eigenvalues, such that the entry Σ_{ii} is the eigenvalue

of the left eigenvector in the i -th column of \mathbf{V} . Normally the inverse of a matrix could be written as $\mathbf{V}\Sigma^{-1}\mathbf{V}^{-1}$ where for a diagonal matrix $[\Sigma^{-1}]_{ii} = 1/\Sigma_{ii}$ but we know that the Laplacian has one zero eigenvector and then this inverse is not defined. So the pseudo-inverse defines $[\tilde{\Sigma}^{-1}]_{ii} = 1/\Sigma_{ii}$ except when $\Sigma_{ii} = 0$ when we simply leave $[\tilde{\Sigma}^{-1}]_{ii} = 0$. In terms of solving this in practice, see comments in Bacco et al. (2018) about linear methods to find the pseudo-inverse of a Laplacian.

We also note that for any node with just one edge (in any direction) such dangling nodes have trivial solutions where their coordinate is exactly $R = 1.0$ more or exactly $R = 1.0$ less than the node they are connected to, depending on the direction of the edge. Further, if a node is part of a line, where every node in the line except at the two ends, has exactly one edge out and one edge in, we can also deduce the rating of the nodes on the line from the values of the ratings of the nodes at the end of the line. In both cases these nodes can be eliminated to reduce the size of the Laplacian matrix we need to solve. In practice these simplifications are unlikely to have much effect on the overall time to find a solution.

For a data set that is approximately a DAG, the rating values r_i define a total order on the nodes. From that we can now define a DAG with adjacency matrix \mathbf{D} as

$$D_{ij} = (A_{ij} + A_{ji}) \Theta(r_i - r_j), \quad (6.6.16)$$

$$\text{where } \Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (6.6.17)$$

(T) What is interesting is if we started with a DAG it is not clear if we get the same DAG back through this process. Seems like a weakness of the solution. The reason for this is that the only information on the asymmetry is in the \mathbf{J} vector of (6.6.10) which is simply a function of the degree of each node, a local not a global measure. (end T)

Hierarchies and DAGs from Arbitrary Graphs

(T) This subsection is from DAG notes on Bitbucket but also see removing cycles section in the same notes. (end T)

Direction from Node Centrality

This may be from Sugiyama et al. (1981) as used in (Batty, 2013, ch.5, p.169 and fig 5.16).

1. Take an undirected graph with no self-loops, adjacency matrix $A_{ij} = A_{ji}$, $A_{ii} = 1$. In fact self-loops don't effect the result below.
2. Calculate a node centrality measure, c_i .
3. Create a DAG such that the edge order is always from high to low centrality measure, that is $D_{ij} = 1$ if $c_j > c_i$ and $A_{ij} \neq 0$, otherwise $D_{ij} = 0$

$$D_{ij} = \frac{1}{2} (A_{ij} + A_{ji}) \Theta(c_i - c_j), \quad \text{where } \Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (6.6.18)$$

Other Hierarchical Methods

1. Any method that makes a DAG from a directed graph, e.g. agony "Hierarchies in directed networks" Tatti (2015) and "Tiers for peers: a practical algorithm for discovering hierarchy in weighted networks" Tatti (2017).
2. "Finding hierarchy in directed online social networks" Gupte et al. (2011).

The AGONY method (Gupte et al., 2011, Tatti, 2017) produces a node rating r_u of node u . The agony node vector \mathbf{r} is found by finding the node vector \mathbf{w} that minimises the agony $Q(\mathcal{G}; \mathbf{w})$ of a network \mathcal{G} which is the sum of the agony $\Theta(w_u - w_v + 1, 0)$ of each edge (u, v) . That is^{(T)⁶}

$$Q(\mathcal{G}; \mathbf{r}) = \min_{\mathbf{w}} (Q(\mathcal{G}; \mathbf{w})), \quad (6.6.19)$$

$$Q(\mathcal{G}; \mathbf{w}) = \sum_{(u,v) \in \mathcal{E}} \Theta(w_u - w_v + 1, 0). \quad (6.6.20)$$

Note a non-zero contribution comes from any edge where the node rating does not reflect the edge direction by a factor of one.^{(T)⁷ Note this is similar to SpringRank in the sense that we are looking at the L_p metric (Minkowski distance) to define the difference in the rating of two nodes connected by an edge, but with a subtle difference}

$$\left(\sum_{(u,v) \in \mathcal{E}} |w_u - w_v + 1|^p \right)^{1/p}, \quad \left(\sum_{(u,v) \in \mathcal{E}} \Theta(w_u - w_v + 1)^p \right)^{1/p}, \quad (6.6.21)$$

6.7 Hierarchies, Trees, Hyperbolic Space and DAGs

Hierarchies, which are naturally thought of as rooted directed trees, are a special case of DAGs. Some of the examples of DAGs, such as code dependencies are natural examples of DAGs. These are based on the rigorous proof any weighted tree can be embedded in a hyperbolic space while preserving their metric (Gromov, 1987, Bowditch, 2005, Sarkar, 2011a,b, Suzuki et al., 2019) (see also Sugiyama et al. (1981)?).

⁶(T) Is the edge direction convention correct here?

⁷(T) Note a non-zero contribution comes from any edge where the node rating does not reflect the edge direction by a factor of one. It is a bit like SpringRank but using a L_p metric of $p = 1$ but also reminds us of the different node activation functions in neural networks such as ReLu.

Acknowledgements

Thanks to ? for useful conversations.

Bibliography

- G. Agarwal and D. Kempe. Modularity-maximizing network communities via mathematical programming. *Eur. Phys. J. B*, 66:409–418, 2008. doi: 10.1140/epjb/e2008-00425-1. URL <http://www.citebase.org/abstract?id=oai:arXiv.org:0710.2533>.
- M. H. Albert and A. M. Frieze. Random graph orders. *Order*, 6(1):19–30, 1989. URL <http://link.springer.com/article/10.1007/BF00341633>.
- E. Alessandri. Identifying technological trajectories in the mining sector using patent citation networks. *Resources Policy*, 80:103130, jan 2023. doi: 10.1016/j.resourpol.2022.103130.
- A. Argent-Katwala, T. Evans, and U. Harder. Exploration of the network spun by website users. In *Proceedings of the 23rd Annual UK Performance Engineering Workshop (UKPEW 2007), Edge Hill University, UK, July 9th-10th, 2007*, 2007.
- S. Arya, M. J. Golin, and K. Mehlhorn. On the expected depth of random circuits. *Combinatorics, Probability and Computing*, 8(3):209–228, May 1999. ISSN 1469-2163. doi: 10.1017/s096354839900382x.
- I. Asimov. *The Genetic Code*. The Orion Press, 1962.
- S. Babul, K. Devriendt, and R. Lambiotte. Gromov centrality: A multi-scale measure of network centrality using triangle inequality excess. *Phys. Rev. E*, 106:034312, May 2022. doi: 10.1103/PhysRevE.106.034312.
- C. D. Bacco, D. B. Larremore, and C. Moore. A physical model for efficient ranking in networks. *Science Advances*, 4:eaar8260, Sept. 2018. doi: 10.1126/sciadv.aar8260.
- A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:173, 1999.
- N. Barbieri, C. Ghisetti, M. Gilli, G. Marin, and F. Nicolli. A survey of the literature on environmental innovation based on main path analysis. In S. Z. Brian Chiang Lin, editor, *Environmental Economics and Sustainability*, pages 221–250. Wiley Online Library, 2017. ISBN 9781119328209. doi: 10.1002/9781119328223.ch10. URL <http://www.sustainability-seeds.org/papers/RePec/srt/wpaper/0715.pdf>.
- M. Barthélémy. Transitions in spatial networks. *Comptes Rendus Physique*, 4:205–232, Oct. 2018. doi: 10.1016/j.crhy.2018.10.006. URL <http://arxiv.org/abs/1810.08764v2>.
- V. Batagelj. Efficient algorithms for citation network analysis. Technical report, University of Ljubljana, 2003. URL <https://arxiv.org/abs/cs/0309023>.
- V. Batagelj, P. Doreian, A. Ferligoj, and N. Kejzár. *Understanding Large Temporal Networks and Spatial Networks: Exploration, Pattern Searching, Visualization and Network Evolution*. John Wiley & Sons Ltd, October 2014. ISBN 9781118915370. doi: 10.1002/9781118915370.

- URL <http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470714522.html>. 464 pages, ISBN: 978-0-470-71452-2.
- M. Batty. *The New Science of Cities*. MIT Press. MIT Press, 2013. ISBN 9780262019521.
 URL <https://books.google.co.uk/books?id=yX-YAQAAQBAJ>.
- A. Bavelas. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.
- A. Békéssy, B. P., and J. Komlos. Asymptotic enumeration of regular matrices. *Stud. Sci. Math. Hungar.*, 7:343–353, 1972.
- C. Bender and S. Orszag. *Advanced Mathematical Methods for Scientists and Engineers*. Springer, 1999.
- E. A. Bender and E. R. Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A*, 24(3):296–307, 1978. doi: 10.1016/0097-3165(78)90059-6.
- M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.
- A. Bevan. Political geography and palatial crete. *Journal of Mediterranean Archaeology*, 23(1):27–54, 2010. ISSN 17431700. doi: 10.1558/jmea.v23i1.27. URL <http://www.equinoxjournals.com/JMA/article/view/9217>.
- G. Bianconi and A.-L. Barabási. Competition and multiscaling in evolving networks. *EPL (Europhysics Letters)*, 54(4):436–, 2001. ISSN 0295-5075. URL <http://stacks.iop.org/0295-5075/54/i=4/a=436>.
- M. Boguñá, F. Papadopoulos, and D. Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(6):62, Sep 2010. ISSN 2041-1723. doi: 10.1038/ncomms1063. URL <http://dx.doi.org/10.1038/ncomms1063>.
- B. Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European J. Combin.*, 1:311–316, 1980.
- B. Bollobás and G. Brightwell. Box-spaces and random partial orders. *Transactions of the American Mathematical Society*, 324:59–72, 1991. URL <http://www.jstor.org/stable/2001495>.
- S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca. Network analysis in the social sciences. *Science*, 323(5916):892–895, Feb. 2009. URL <http://www.sciencemag.org/content/323/5916/892.abstract>.
- B. H. Bowditch. A course on geometric group theory. Technical report, U. Southampton, 2005. URL <https://www.math.ucdavis.edu/~kapovich/280-2009/bhb-ggtcourse.pdf>.
- U. Brandes, G. Robins, A. McCranie, and S. Wasserman. What is network science? *Net Sci*, 1(01):1–15, Apr 2013. ISSN 2050-1250. doi: 10.1017/nws.2013.2. URL <http://dx.doi.org/10.1017/nws.2013.2>.
- N. Broutin and O. Fawzi. Longest path distance in random circuits. *Combinatorics, Probability and Computing*, 21(6):856–881, July 2012. ISSN 1469-2163. doi: 10.1017/s0963548312000260.

- R. Brüggemann, B. Münzer, and E. Halfon. An algebraic/graphical tool to compare ecosystems with respect to their pollution – the German river “Elbe” as an example - I: hasse-diagrams. *Chemosphere*, 28(5):863–872, 1994a.
- R. Brüggemann, K. Voigt, and B. Münzer. The technique of hasse diagrams applied on environmental online databanks. *Chemosphere*, 29(4):683–691, 1994b.
- R. Bruggemann, E. Halfon, G. Welzl, K. Voigt, and C. Steinberg. Applying the concept of partially ordered sets on the ranking of near-shore sediments by a battery of tests. *J. Chem. Inf. Model.*, 41(4):918–925, Jul 2001. ISSN 1549-9596. doi: 10.1021/ci000055k. URL <http://dx.doi.org/10.1021/ci000055k>.
- T. Brughmans and M. A. Peeples. *Network Science in Archaeology*. Cambridge Manuals in Archaeology. Cambridge University Press, Apr. 2023. ISBN 9781009170642. doi: 10.1017/9781009170659.
- D. S. Callaway, J. E. Hopcroft, J. M. Kleinberg, M. E. J. Newman, and S. H. Strogatz. Are randomly grown graphs really random? *Phys. Rev. E*, 64(4):041902, Sep 2001. doi: 10.1103/PhysRevE.64.041902.
- K. M. Carley, N. P. Hummon, and M. Harty. Scientific influence: An analysis of the main path structure in the journal of conflict resolution. *Science Communication*, 14(4):417–447, Jun 1993. ISSN 1075-5470. doi: 10.1177/107554709301400406. URL <http://dx.doi.org/10.1177/107554709301400406>.
- C. Carstens. A uniform random graph model for directed acyclic networks and its effect on motif-finding. *Journal of Complex Networks*, page cnu028, 2014. doi: 10.1093/comnet/cnu028. URL <http://comnet.oxfordjournals.org/content/early/2014/06/27/comnet.cnu028.short>.
- M. Cerinsek and V. Batagelj. Network analysis of zentralblatt math data. *Scientometrics*, 102(1):977–1001, Sep 2014. ISSN 1588-2861. doi: 10.1007/s11192-014-1419-z. URL <http://dx.doi.org/10.1007/s11192-014-1419-z>.
- S. Chamberlain. Consuming article-level metrics: Observations and lessons. *Information Standards Quarterly*, 25(2):4, Jun 2013. doi: 10.3789/isqv25no2.2013.02. URL <http://dx.doi.org/10.3789/isqv25no2.2013.02>.
- B. Chen, Z. Lin, and T. S. Evans. Analysis of the wikipedia network of mathematicians. Technical report, Imperial College London, 2017. URL <http://hdl.handle.net/10044/1/77436>.
- L. Chuan Wen. *On The Topology Of Network Fine Structures*. PhD thesis, Imperial College London, 2015.
- F. Chung and L. Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, May 2001. doi: 10.1006/aama.2001.0720.
- F. Chung and L. Lu. The average distance in a random graph with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, jan 2002a. doi: 10.1073/pnas.154279599.
- F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002b.

- F. R. Chung and L. Lu. *Complex graphs and networks*. Number 107 in Regional Conference Series in Mathematics. American Mathematical Soc., 2006.
- A. Clauset. Network analysis and modeling lecture notes. Technical report, University of Colorado at Boulder, 2013. URL <http://tuvalu.santafe.edu/~aaronc/courses/5352/>.
- J. Clough and T. Evans. What is the dimension of citation space? Technical report, Imperial College London, 2014. URL <http://arxiv.org/abs/1408.1274>.
- J. R. Clough and T. S. Evans. Embedding graphs in lorentzian spacetime. Technical report, Imperial College London, 2016. URL <http://arxiv.org/abs/1602.03103>.
- J. R. Clough, J. Gollings, T. V. Loach, and T. S. Evans. Transitive reduction of citation networks. *Journal of Complex Networks*, 2014. doi: 10.1093/comnet/cnu039.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, volume 2. MIT press Cambridge, 3rd edition, Sept. 2009. ISBN 0262033844. URL https://www.ebook.de/de/product/8474892/thomas_h_cormen_charles_e_leiserson_ronald_l_rivest_clifford_stern_introduction_to_algorithms.html.
- M. Coscia. *The Atlas for the Aspiring Network Scientist*. Michele Coscia, Jan. 2021. ISBN 978-8797282403.
- J. Courtiel and M. Pépin. Random generation of git graphs. In S. Brlek and L. Ferrari; editors, *GASCom 2024*, volume 403 of *Electronic Proceedings in Theoretical Computer Science*, pages 79–86. arXiv, Mar. 2024. doi: 10.4204/EPTCS.403.18.
- S. Cunningham. *Causal Inference : The Mixtape*. Yale University Press, 2021.
- W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Structural Analysis in the Social Sciences (No. 27). Cambridge University Press, 2005. doi: 10.2277/0521602629.
- M. M. Deza and E. Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009.
- J. Díaz, D. Mitsche, G. Perarnau, and X. Pérez-Giménez. On the relation between graph distance and euclidean distance in random geometric graphs. *Advances in Applied Probability*, 48(3):848–864, 2016. doi: <https://doi.org/10.1017/apr.2016.31>. URL <https://arxiv.org/abs/1404.4757>.
- R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. ISSN 0003486X. URL <http://www.jstor.org/stable/1969503>.
- P. Doreian and A. Mrvar. Partitioning signed social networks. *Social Networks*, 31:1–11, 2009. doi: 10.1016/j.socnet.2008.08.001.
- S. N. Dorogovtsev, J. F. F. Mendes, and A. Samukhin. Principles of statistical mechanics of uncorrelated random networks. *Nuclear Physics B*, 666(3):396–416, Sept. 2003. doi: [10.1016/S0550-3213\(03\)00504-2](https://doi.org/10.1016/S0550-3213(03)00504-2); 2396.
- F. Dowker. Causal sets as discrete spacetime. *Contemporary Physics*, 47(1):1–9, 2006. doi: 10.1080/17445760500356833. URL <http://tandfprod.literatumonline.com/doi/abs/10.1080/17445760500356833>.

- P. G. Doyle and J. L. Snell. *Random Walks and Electric Networks*, volume 22 of *The Carus Mathematical Monographs*. Mathematical Association of America, Washington DC, 1984. ISBN 978-0-88385-024-4. doi: 10.5948/UPO9781614440222.
- R. Durrett. *Random Graph Dynamics*. CUP, 2006. ISBN 978-0-521-86656-9. URL <http://web.archive.org/web/20060205193451/http://www.math.cornell.edu/~durrett/RGD/RGD.html>.
- D. Eck. *Introduction to programming using Java*. Hobart and William Smith Colleges, Department of mathematics and computer science, Geneva (NY, 8.1.2 edition, 2007. ISBN 9781616100995. URL <http://math.hws.edu/javanotes/>.
- A. Edalat and R. Heckmann. A computational model for metric spaces. *Theoretical Computer Science*, 193(1-2):53–73, Feb. 1998. doi: 10.1016/s0304-3975(96)00243-5.
- R. B. Ellis, J. L. Martin, and C. Yan. Random geometric graph diameter in the unit ball. *Algorithmica*, 47(4):421–438, Jan. 2007. doi: 10.1007/s00453-006-0172-y.
- F. Elwert. Graphical causal models. In S. L. Morgan, editor, *Handbook of Causal Analysis for Social Research*, pages 245–273. Springer Netherlands, 2013. ISBN 9789400760943. doi: 10.1007/978-94-007-6094-3_13.
- P. Erdős and A. Rényi. On random graphs. i. *Publicationes Mathematicae*, 6:290–297, 1959. URL http://www.renyi.hu/~p_erdos/1959-11.pdf.
- L. Esperet and G. Joret. Boxicity of graphs on surfaces. *Graphs and Combinatorics*, 29(3):417–427, Jan 2012. ISSN 1435-5914. doi: 10.1007/s00373-012-1130-x. URL <http://dx.doi.org/10.1007/s00373-012-1130-x>.
- E. Estrada. Characterization of topological keystone species. *Ecological Complexity*, 4(1–2):48–57, Mar. 2007. ISSN 1476-945X. doi: 10.1016/j.ecocom.2007.02.018.
- T. Evans, L. Calmon, and V. Vasiliauskaitė. The longest path in the price model. *Scientific Reports*, 10(1):10503, 2020. doi: 10.1038/s41598-020-67421-8.
- T. S. Evans and B. Chen. Linking the network centrality measures closeness and degree. *Communications Physics*, 5:172, 2022. doi: 10.1038/s42005-022-00949-5. URL <https://rdcu.be/cQRgW>.
- D. Fasino, A. Tonetto, and F. Tudisco. Generating large scale-free networks with the chung-lu random graph model. *Networks*, 78(2):174–187, Dec. 2020. ISSN 1097-0037. doi: 10.1002/net.22012.
- A. Fedor and V. Vasas. The robustness of keystone indices in food webs. *Journal of Theoretical Biology*, 260(3):372–378, Oct. 2009. ISSN 0022-5193. doi: 10.1016/j.jtbi.2009.07.003.
- F. Filippin. Do main paths reflect technological trajectories? applying main path analysis to the semiconductor manufacturing industry. *Scientometrics*, 126(8):6443–6477, June 2021. doi: 10.1007/s11192-021-04023-9.
- R. Fontana, A. Nuvolari, and B. Verspagen. Mapping technological trajectories as patent citation networks. an application to data communication standards. *Economics of Innovation and New Technology*, 18(4):311–336, Jun 2009. ISSN 1476-8364. doi: 10.1080/10438590801969073. URL <http://dx.doi.org/10.1080/10438590801969073>.

- L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.
- L. C. Freeman. The gatekeeper, pair-dependency and structural centrality. *Quality and Quantity*, 14(4), Aug. 1980. ISSN 1573-7845. doi: 10.1007/bf00184720.
- L. C. Freeman. Spheres, cubes and boxes: graph dimensionality and network structure. *Social Networks*, 5(2):139–156, 1983.
- L. C. Freeman. *The Development Of Social Network Analysis: A Study In The Sociology Of Science*. ΣP Empirical Press, Vancouver, 2004. ISBN 1-59457-714-5. URL <https://www.researchgate.net/publication/239228599>.
- L. K. Gallos, C. Song, and H. A. Makse. A review of fractality and self-similarity in complex networks. *Physica A: Statistical Mechanics and its Applications*, 386(2):686–691, Dec 2007. ISSN 0378-4371. doi: 10.1016/j.physa.2007.07.069. URL <http://dx.doi.org/10.1016/j.physa.2007.07.069>.
- E. Garfield, I. H. Sher, and R. J. Torpie. *The use of citation data in writing the history of science*. Institute for Scientific Information Inc. Philadelphia. Pennsylvania, USA, 1964. URL <http://www.garfield.library.upenn.edu/papers/useofcitdatawritinghistofsci.pdf>.
- D. Gascuel, S. Guénette, and D. Pauly. The trophic-level-based ecosystem modelling approach: theoretical overview and practical uses. *ICES Journal of Marine Science*, 68(7):1403–1416, May 2011. ISSN 1054-3139. doi: 10.1093/icesjms/fsr062.
- J. Gehrke, P. Ginsparg, and J. Kleinberg. Overview of the 2003 kdd cup. *ACM SIGKDD Explorations Newsletter*, 5(2):149–151, 2003.
- E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, Dec. 1959. doi: 10.1214/aoms/1177706098.
- I. Gradshteyn and I. Ryzhik. *Table of Integrals, Series and Products*. Academic Press Inc., 2rev ed edition edition, 1980.
- S. Graham, I. Milligan, and S. Weingart. *Exploring Big Historical Data: The Historian's Macroscope*. Imperial College Press, 2015.
- S. Greenland, J. Pearl, and M. Robins, James. Causal diagrams for epidemiologic research. *Epidemiology*, 10(1):37–48, 1999. URL <https://www.jstor.org/stable/3702180>.
- M. Gromov. Hyperbolic groups. In *Essays in Group Theory*, pages 75–263. Springer New York, 1987. doi: 10.1007/978-1-4613-9586-7_3.
- D. Grote. The myth of performance metrics. *Harvard Business Review*, 09 2011.
- S. Gualdi, M. Medo, and Y.-C. Zhang. Influence, originality and similarity in directed acyclic graphs. *EPL (Europhysics Letters)*, 96(1):18004, Sep 2011. ISSN 1286-4854. doi: 10.1209/0295-5075/96/18004. URL <http://dx.doi.org/10.1209/0295-5075/96/18004>.
- J. Guare. *Six Degrees of Separation: A Play*. New York: Random House, 1990.
- P. R. Guimaraes Jr, M. A. De Aguiar, J. Bascompte, P. Jordano, and S. F. Dos Reis. Random initial condition in small barabasi-albert networks and deviations from the scale-free behavior. *Physical Review E*, 71(3):037101, 2005.

- M. Gupte, P. Shankar, J. Li, S. Muthukrishnan, and L. Iftode. Finding hierarchy in directed online social networks. In *Proceedings of the 20th international conference on World wide web*, pages 557–566. ACM, 2011. doi: 10.1145/1963405.1963484.
- B. H. Hall, A. B. Jaffe, and M. Trajtenberg. The nber patent citation data file: Lessons, insights and methodological tools. Technical report, National Bureau of Economic Research, 2001. URL <http://www.nber.org/papers/w8498>.
- F. Harary. Status and contrastatus. *Sociometry*, 22(1):23, Mar. 1959. doi: 10.2307/2785610.
- F. Harary. Who eats whom. *General Systems*, 6:41–44, 1961.
- S. Haustein, T. D. Bowman, and R. Costas. When is an article actually published? an analysis of online availability, publication, and indexation dates. In *Proceedings of the 15th International Society of Scientometrics and Informetrics Conference*, pages 1170–1179, 2015.
- P. Healy and N. S. Nikolov. How to layer a directed acyclic graph. In *International Symposium on Graph Drawing*, pages 16–30. Springer, 2001.
- M. Ho, H. Price, T. Evans, and E. O’Sullivan. Dynamics of technology emergence in innovation networks. *Sci Rep*, 14:1411, 2024a. doi: 10.1038/s41598-023-50280-4. URL <http://dx.doi.org/10.1038/s41598-023-50280-4>.
- M. Ho, H. C. Price, T. S. Evans, and E. O’Sullivan. Who made the mrna vaccine? measuring division of labour in therapeutic innovation. *Nature Biotechnology*, in press, 2024b.
- P. Holme. Modern temporal network theory: A colloquium. *The European Physical Journal B*, 88:1–30, Aug. 2015. doi: 10.1140/epjb/e2015-60657-4.
- P. Holme and J. Saramäki. Temporal Networks. *Physics Reports*, 519:97–125, Aug. 2012. doi: 10.1016/j.physrep.2012.03.001. URL <http://arXiv.org/abs/1108.1780>.
- N. P. Hummon and P. Doreian. Connectivity in a citation network: The development of dna theory. *Social Networks*, 11(1):39–63, 1989. URL <http://www.sciencedirect.com/science/article/pii/0378873389900178>.
- R. Ilie, G. B. Thompson, and D. D. Reid. A numerical study of the correspondence between paths in a causal set and geodesics in the continuum. *Classical and Quantum Gravity*, 23(10):3275–3285, May 2006. doi: 10.1088/0264-9381/23/10/002.
- S. Janson. The number of descendants in a random directed acyclic graph. *Random Structures & Algorithms*, 64(3):768–803, Nov. 2023. ISSN 1098-2418. doi: 10.1002/rsa.21195.
- H. Jennings. Structure of leadership-development and sphere of influence. *Sociometry*, 1(1/2): 99, July 1937. ISSN 0038-0431. doi: 10.2307/2785262.
- F. Jordan, A. Takacs-Santa, and I. Molnar. A reliability theoretical quest for keystones. *Oikos*, 86(3):453, Sept. 1999. ISSN 0030-1299. doi: 10.2307/3546650.
- F. Jordán and I. Scheuring. Network ecology: topological constraints on ecosystem dynamics. *Physics of Life Reviews*, 1(3):139–172, Dec. 2004. ISSN 1571-0645. doi: 10.1016/j.plrev.2004.08.001.
- A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, Nov. 1962. ISSN 1557-7317. doi: 10.1145/368996.369025.

- M. J. Kalutkiewicz and R. L. Ehman. Patents as proxies: Nih hubs of innovation. *Nature Biotechnology*, 32(6):536–537, June 2014. ISSN 1546-1696. doi: 10.1038/nbt.2917.
- B. Karrer and M. E. J. Newman. Random acyclic networks. *Phys. Rev. Lett.*, 102:128701, 2009. doi: 10.1103/PhysRevLett.102.128701. URL <http://uk.arxiv.org/abs/0902.4013>.
- A. P. Kartun-Giles, M. Barthelemy, and C. P. Dettmann. Shape of shortest paths in random spatial networks. *Phys. Rev. E*, 100:032315, Sep 2019. doi: 10.1103/PhysRevE.100.032315. URL <https://link.aps.org/doi/10.1103/PhysRevE.100.032315>.
- KDD cup. Data for kdd cup. <https://www.kdd.org/kdd-cup/view/kdd-cup-2003/Data>, 2003. URL <http://www.cs.cornell.edu/projects/kddcup/datasets.html>.
- E. H. Kim, Y. K. Jeong, Y. Kim, and M. Song. Exploring scientific trajectories of a large-scale dataset using topic-integrated path extraction. *Journal of Informetrics*, 16(1):101242, Feb. 2022. doi: 10.1016/j.joi.2021.101242.
- M. Kitromilidis and T. S. Evans. Community detection with metadata in a network of biographies of western art painters. Technical report, Imperial College London, 2017. URL <http://hdl.handle.net/10044/1/77437>.
- J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, Sept. 1999. ISSN 1557-735X. doi: 10.1145/324133.324140.
- D. Krioukov, F. Papadopoulos, A. Vahdat, and M. Boguna. Curvature and temperature of complex networks. *Phys. Rev. E*, 80,:035101(2009), Mar. 2009. doi: 10.1103/PhysRevE.80.035101.
- C.-H. Kuan. Does main path analysis prefer longer paths? *Scientometrics*, 128(1):841–851, oct 2022. doi: 10.1007/s11192-022-04543-y.
- A. N. Langville and C. D. Meyer. *Google’s PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011. ISBN 9780691152660.
- E. Letizia, P. Barucca, and F. Lillo. Resolution of ranking hierarchies in directed networks. *PLOS ONE*, 13(2):e0191604, feb 2018. doi: 10.1371/journal.pone.0191604.
- S. Libralato, V. Christensen, and D. Pauly. A method for identifying keystone species in food web models. *Ecological Modelling*, 195(3–4):153–171, June 2006. ISSN 0304-3800. doi: 10.1016/j.ecolmodel.2005.11.029.
- J. S. Liu and C.-H. Kuan. A new approach for main path analysis: Decay in knowledge diffusion. *Journal of the Association for Information Science and Technology*, 2015. doi: 10.1002/asi.23384.
- J. S. Liu and L. Y. Lu. An integrated approach for main path analysis: Development of the hirsch index as an example. *Journal of the American Society for Information Science and Technology*, 63(3):528–542, 2012. ISSN 1532-2890. doi: 10.1002/asi.21692. URL <http://dx.doi.org/10.1002/asi.21692>.
- J. S. Liu, L. Y. Y. Lu, and M. H.-C. Ho. A few notes on main path analysis. *Scientometrics*, 119(1):379–391, Feb. 2019. ISSN 1588-2861. doi: 10.1007/s11192-019-03034-x.
- D. Lucio-Arias and L. Leydesdorff. Main-path analysis and path-dependent transitions in histcite-based historiograms. *J. Am. Soc. Inf. Sci.*, 59(12):1948–1962, Oct 2008. ISSN 1532-2890. doi: 10.1002/asi.20903. URL <http://dx.doi.org/10.1002/asi.20903>.

- E. Malmi, N. Tatti, and A. Gionis. Beyond rankings: comparing directed acyclic graphs. *Data Mining and Knowledge Discovery*, 29(5):1233–1257, Mar. 2015. ISSN 1573-756X. doi: 10.1007/s10618-015-0406-1.
- S. Maslov and K. Sneppen. Specificity and stability in topology of protein networks. *Science*, 296:910, 2002. doi: 10.1126/science.1065103. URL <http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0205380>.
- N. Masuda and R. Lambiotte. *Temporal Networks*. Imperial College Press World Scientific, 2016.
- B. D. McKay, F. E. Oggier, G. F. Royle, N. J. A. Sloane, I. M. Wanless, and H. S. Wilf. Acyclic digraphs and eigenvalues of (0,1)-matrices. *J. Integer Sequences*, 7(2):1–5, Oct. 2004. URL <http://arxiv.org/abs/math/0310423>.
- R. K. Merton. The matthew effect in science: The reward and communication systems of science are considered. *Science*, 159(3810):56–63, 1968. doi: 10.1126/science.159.3810.56.
- R. K. Merton. The matthew effect in science, ii: Cumulative advantage and the symbolism of intellectual property. *isis*, 79(4):606–623, 1988.
- S. Milgram. The small world problem. *Psychology Today*, 1967.
- L. Mirsky. A dual of dilworth’s decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971. ISSN 00029890, 19300972. URL <http://www.jstor.org/stable/2316481>.
- H. Moed, R. De Bruin, and T. Van Leeuwen. New bibliometric tools for the assessment of national research performance: Database description, overview of indicators and first applications. *Scientometrics*, 33(3):381–422, July 1995. URL <http://dx.doi.org/10.1007/BF02017338>.
- M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6:161–180, 1995. URL citeseer.ist.psu.edu/molloy95critical.html.
- S. L. Morgan and C. Winship. *Counterfactuals and causal inference*. Analytical methods for social research. Cambridge University Press, New York, NY, second edition edition, 2015. ISBN 9781107694163. Title from publisher’s bibliographic system (viewed on 05 Oct 2015).
- B. Münzer, R. Brüggemann, and E. Halfon. An algebraic/graphical tool to compare ecosystems with respect to their pollution ii: Comparative regional analysis. *Chemosphere*, 28(5):873–879, 1994.
- F. Narin, K. S. Hamilton, and D. Olivastro. The increasing linkage between us technology and public science. *Research policy*, 26(3):317–330, 1997.
- M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64(2):026118, Jul 2001. doi: 10.1103/PhysRevE.64.026118.
- J. Pearl. Causal diagrams for empirical research. *Biometrika*, 2(4):669–688, 1995. URL <https://www.jstor.org/stable/2337329>.
- M. Penrose. *Random Geometric Graphs*. OUP, 2003.

- S. Pimm and J. H. Lawton. On feeding on more than one trophic level. *Nature*, 275(5680): 542–544, 1978. doi: 10.1038/275542a0.
- S. L. Pimm. *Food Webs*. Population and Community Biology. Springer, Dordrecht, 1982. ISBN 9789400959255.
- A. J. Pitts and C. R. Fowler. Comparison of open-source software for producing directed acyclic graphs. *Journal of Causal Inference*, 12(1), Jan. 2024. ISSN 2193-3685. doi: 10.1515/jci-2023-0031.
- D. J. d. S. Price. The scientific foundations of science policy. *Nature*, 206:233–238, 1965a.
- D. S. Price. Networks of scientific papers. *Science*, 149:510–515, 1965b. URL <http://garfield.library.upenn.edu/papers/pricenetworks1965.pdf>.
- D. S. Price. A general theory of bibliometric and other cumulative advantage processes. *J.Amer.Soc.Inform.Sci.*, 27(5):292–306, 1976. ISSN 0002-8231. doi: 10.1002/asi.4630270505. URL <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.4630270505>.
- N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, Jan. 2007. doi: 10.1093/bioinformatics/btl301. URL <http://bioinformatics.oxfordjournals.org/content/23/2/e177.abstract>.
- A. Raveh and S. F. Landau. Partial order scalogram analysis with base coordinates (posac): Its application to crime patterns in all the states in the united states. *Journal of Quantitative Criminology*, 9(1):83–99, March 1993. doi: 10.1007/BF01064238. URL <http://link.springer.com/article/10.1007%2FBF01064238>. 83-99.
- C. Renfrew and E. Level. Exploring dominance: predicting polities from centres. In A. Renfrew and K. Cooke, editors, *Transformations: Mathematical Approaches to Culture Change*, pages 145–67. Academic Press, London, 1979.
- D. Rideout and P. Wallden. Spacelike distance from discrete causal order. *Classical and Quantum Gravity*, 26(15):155013–, AUG 7 2009. ISSN 0264-9381. doi: 10.1088/0264-9381/26/15/155013. URL <http://stacks.iop.org/0264-9381/26/i=15/a=155013>.
- R. W. Robinson. Counting labeled acyclic digraphs. In F. Harary, editor, *New directions in the theory of graphs*, pages 239–273. Ann Arbor Conference on Graph Theory (3rd : 1971 : University of Michigan), Academic Press, 1973. ISBN 012324255X.
- J. M. Rohrer. Thinking clearly about correlations and causation: Graphical causal models for observational data. *Advances in Methods and Practices in Psychological Science*, 1(1):27–42, Jan. 2018. ISSN 2515-2467. doi: 10.1177/2515245917745629.
- R. Sarkar. Low distortion delaunay embedding of trees in hyperbolic plane. In *International Symposium on Graph Drawing*, pages 355–366. Springer, 2011a. doi: 10.1007/978-3-642-25878-7_34.
- R. Sarkar. Low distortion delaunay embedding of trees in hyperbolic plane, 2011b.
- M. . A. Serrano, D. Krioukov, and M. Boguñá. Self-similarity of complex networks and hidden metric spaces. *Physical Review Letters*, 100(7), Feb 2008. ISSN 1079-7114. doi: 10.1103/physrevlett.100.078701. URL <http://dx.doi.org/10.1103/PhysRevLett.100.078701>.
- S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31, 2002.

- D. Shi, F. Shang, B. Chen, P. Expert, L. Lü, H. E. Stanley, R. Lambiotte, T. S. Evans, and R. Li. Local dominance unveils clusters in networks. *Communications Physics*, Sept. 2024. doi: 10.48550/ARXIV.2209.15497. URL <http://hdl.handle.net/10044/1/104821>.
- H. Simon. On a class of skew distribution functions. *Biometrika*, 42:425, 1955.
- G. Smith. *Introductory Mathematics: Algebra and Analysis*. Springer Undergraduate Mathematics Series. Springer London, 1998. ISBN 9783540761785. URL <https://books.google.co.uk/books?id=MBq1tPT6F74C>.
- Solomonoff and Rapoport. Connectivity of random nets. *The Bulletin of Mathematical Biophysics*, 1951.
- C. Song, L. K. Gallos, S. Havlin, and H. A. Makse. How to calculate the fractal dimension of a complex network: the box covering algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 03:P03006, Mar 2007. ISSN 1742-5468. doi: 10.1088/1742-5468/2007/03/p03006. URL <http://dx.doi.org/10.1088/1742-5468/2007/03/P03006>.
- K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi: 10.1109/TSMC.1981.4308636.
- S. Surya. The causal set approach to quantum gravity. *Living Reviews in Relativity*, 22(1), sep 2019. doi: 10.1007/s41114-019-0023-1.
- R. Suzuki, R. Takahama, and S. Onoda. Hyperbolic disk embeddings for directed acyclic graphs. In *Proceedings of the 36th International Conference on MachineLearning, Long Beach, California, PMLR 97*, 2019, Feb. 2019.
- D. Tang, F. Chen, C. De Leon, T. Wattanawaroon, J. Yun, S. Seshadri, and A. G. Parameswaran. Efficient and compact spreadsheet formula graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2634–2646. IEEE, 2023. doi: 10.1109/ICDE55515.2023.00202.
- N. Tatti. Hierarchies in directed networks. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 991–996. IEEE, 2015.
- N. Tatti. Tiers for peers: a practical algorithm for discovering hierarchy in weighted networks. *Data Mining and Knowledge Discovery*, 31(3):702–738, 2017. doi: 10.1007/s10618-016-0485-7.
- P. W. G. Tennant, E. J. Murray, K. F. Arnold, L. Berrie, M. P. Fox, S. C. Gadd, W. J. Harrison, C. Keeble, L. R. Ranker, J. Textor, G. D. Tomova, M. S. Gilthorpe, and G. T. H. Ellison. Use of directed acyclic graphs (dags) to identify confounders in applied health research: review and recommendations. *International Journal of Epidemiology*, 50(2):620–632, Dec. 2020. ISSN 1464-3685. doi: 10.1093/ije/dyaa213.
- J. Ugwu and P. Cuffe. Towards the use of sink ancestors and source descendants as a clustering method in a directed acyclic graph. In *2023 58th International Universities Power Engineering Conference (UPEC)*. IEEE, Aug. 2023. doi: 10.1109/upec57427.2023.10294858.
- R. Ulanowicz and C. J. Puccia. Mixed trophic impacts in ecosystems. *Coenoses*, pages 7–16, 1990. URL <https://www.jstor.org/stable/43461017>.

- S. Valverde and R. V. Solé. Punctuated equilibrium in the large-scale evolution of programming languages. *Journal of The Royal Society Interface*, 12(107):20150249, June 2015. ISSN 1742-5662. doi: 10.1098/rsif.2015.0249.
- V. Vasas and F. Jordán. Topological keystone species in ecological interaction networks: Considering link quality and non-trophic effects. *Ecological Modelling*, 196(3–4):365–378, July 2006. ISSN 0304-3800. doi: 10.1016/j.ecolmodel.2006.02.024.
- V. Vasiliauskaitė and T. S. Evans. Making communities show respect for order. *Applied Network Science*, 5:15, 2020. doi: 10.1007/s41109-020-00255-5. URL <http://arxiv.org/abs/1908.11818>.
- I. Vendrov, R. Kiros, S. Fidler, and R. Urtasun. Order-embeddings of images and language. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings*, Nov. 2016. URL <http://arxiv.org/abs/1511.06361>.
- B. Verspagen. Mapping Technological Trajectories As Patent Citation Networks: A Study On The History Of Fuel Cell Research. *Advances in Complex Systems*, 10(01):93–115, Mar. 2007. doi: 10.1142/s0219525907000945.
- S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, 1994.
- D. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton & Company, 2003.
- P. Winkler. Random orders. *Order*, 1(4):317–331, 1985a. URL <http://link.springer.com/article/10.1007/BF00582738>.
- P. Winkler. Connectedness and diameter for random orders of fixed dimension. *Order*, 2(2):165–171, 1985b. URL <http://link.springer.com/article/10.1007/BF00334854>.
- N. C. Wormald. The asymptotic connectivity of labelled regular graphs. *Journal of Combinatorial Theory, Series B*, 31(2):156–167, 1981. doi: 10.1016/S0095-8956(81)80021-4.
- Y. Xiao, L. Y. Lu, J. S. Liu, and Z. Zhou. Knowledge diffusion path analysis of data quality literature: A main path analysis. *Journal of Informetrics*, 8(3):594–605, Jul 2014. ISSN 1751-1577. doi: 10.1016/j.joi.2014.05.001. URL <http://dx.doi.org/10.1016/j.joi.2014.05.001>.
- W. Yeo, S. Kim, J.-M. Lee, and J. Kang. Aggregative and stochastic model of main path identification: a case study on graphene. *Scientometrics*, 98(1):633–655, oct 2013. doi: 10.1007/s11192-013-1140-3.
- G. U. Yule. A mathematical theory of evolution based on the conclusions of dr. j.c. willis. *F.R.S. Phil. Trans. B*, 21-87:21–87, 1925. doi: 10.1098/rstb.1925.0002.
- W. Zachary. Information-flow model for conflict and fission in small-groups. *Journal Of Anthropological Research*, 33(4):452—473, 1977.
- I. Zakhlebin and E.-A. Horvat. Rank aggregation for determining importance of nodes in complex networks. 2017.
- D. Zhao and A. Strotmann. *Analysis and Visualization of Citation Networks*. Morgan & Claypool Publishers, February 2015. ISBN 9781608459391. doi: 10.2200/S00624ED1V01Y201501ICR039.

- Q. Zhao and X. Feng. Utilizing citation network structure to predict paper citation counts: A deep learning approach. *Journal of Informetrics*, 16(1):101235, 2022. ISSN 1751-1577. doi: <https://doi.org/10.1016/j.joi.2021.101235>. URL <https://www.sciencedirect.com/science/article/pii/S1751157721001061>.

Appendices

Appendix A

Useful Mathematics

A.1 Linear Algebra

The purpose of this section is not to provide a full course on linear algebra. We will not even nail down the precise mathematical conditions for all the results we mention apply. Rather it serves to define the notation we will use, and remind readers of the core properties we will call upon. We will start, however, by asking a wider question, why do we want linear algebra here?

A.1.1 Why is linear algebra relevant?

In science in general, and network science in particular, the use of matrices and vectors to represent processes is widespread. This is part of the mathematical area known as LINEAR ALGEBRA. The key feature of these tools is that they are used when we have a LINEAR PROCESS, that is when the change in one process is proportional to the change in another. A typical example is when $x_i = \sum_j y_{ij}y_j$ where \mathbf{x} is some set of values we are trying to find given a set of values we already know. The dynamics of this system, the science, is encoded in the values A_{ij} . We can think of the x and y values as entries in vectors \mathbf{x} and \mathbf{y} . For instance x_i could be the value of at the next time step of a value y_i and i is the index of a node in the network, A could be the adjacency matrix. The key property of a linear equation is that if we know two solutions, say $x_i^{(1)}$ and $y_i^{(1)}$, and a second solution, say $x_i^{(2)}$ and $y_i^{(2)}$ one solution, then we immediately have a whole family of solutions, say $c_1x_i^{(1)} + c_2x_i^{(2)}$ and $c_1y_i^{(1)} = c_2y_i^{(2)}$ for any constants c_1 and c_2 . This is tremendously powerful as we can reduce the study of the science to a study of a limited number of solutions. In fact one of the key parts of linear algebra is to express the properties of linear equations in terms of a limited set of eigenvectors which are exactly these key solutions out of which all other solutions are built. It is also relatively easy to use the same tools to deal with equations such as $x_i = a_i + \sum_j y_{ij}y_j$ for some values a_i known again from the science.

However, the problem is that nature is not usually described in terms of linear equations. In the real world, if we had two solutions, a stable forest full of red deer and a stable forest ecosystem full of rabbits, we cannot simply say a forest full of the same number of deer and rabbits would also be stable. Quite the opposite, we know species interact, if nothing else indirectly via the limited supply of grass they both eat. Mathematically, most systems are described by non-linear equations, that is where our x_i values are described by equations that don't just involve y_j but could involve $(y_j)^2$, $\sin(y_j)$ or indeed any thing that is not simply y_j . This fact is good, non-linear terms in equations describe interactions. Without interactions, nothing would happen. Particles, cars, people would pass through each other and through anything else, forever moving in a straight line without ever changing. Mathematically, dealing with interactions is challenging. Non-linear equations are far harder to deal with and there is

no single consistent framework to deal with non-linear equations, each context must find the mathematical tools that are most useful.

So now we have a problem, if the world is not linear, why on earth are linear equations and so linear algebra so useful? The answer lies in the idea of small perturbations. Even if the equations are non-linear, there is almost always a limit of “small deviations” in which we can approximate the non-linear equations in terms of some linear terms. That is if we assume y_i are small we can ignore any $(y_i)^2$ terms, approximate terms involving $\sin(y_i)$ by y_i and so forth, until we have linear equations. Of course, the world is not always full of small perturbations, they are one special case. So we may as well start our investigation of any system by looking at this special case before tackling the more general cases. The big advantage, is the tools of linear algebra are then available and the mathematical problem is far easier. Numerical tools for such cases are correspondingly well developed and widely available.

Sometimes, we choose to invent a process to analyse a network. In such cases, we may as well stick to linear processes to make life easy. The process may have no particular relevance to the real world on the surface. However, the small perturbation approximation shows that linear process maybe a good way to represent one particular limit of the real world. For instance, PageRank is used to analyse the importance of nodes in a network of hyperlinked web pages. Underneath is a linear process described by vectors and matrices. Real people are not non-interacting random walkers as described by the mathematics. Yet, it seems to work very well. In this case, it appears that when there are large numbers of people, their average behaviour may be well captured, perhaps to first order, by linear equations. At least, it is captured well enough to provide a better search engine (google) than others around at the time google emerged.

In fact, one of the interesting things about DAGs that many of the traditional applications of linear algebra to network analysis do not work, at least not in the same way.

- $\mathbf{A}^n = 0$ for $n > H$
- Eigenvalues and eigenvectors.
- Non-negative matrices
- Triangular
- Defective matrix does not have a full set of eigenvectors.
- Nilpotent $\mathbf{A}^n = 0$ for $n > H$ where H is the index (or degree) of the matrix.

A DEFECTIVE MATRIX is a square matrix that does not have a complete basis of eigenvectors, and is therefore not diagonalizable. In particular, an $n \times n$ matrix is defective if and only if it does not have $n \times n$ linearly independent eigenvectors.

A.2 Generating Functions

Generating functions are a powerful mathematical technique and are useful in many situations. A generating function is a formal mathematical expression used to represent an infinite sequence of numbers as the coefficients of a formal power series. The master equation of the Price/BA model can provide an illustration of their use with difference equations for functions of a discrete variable.

A useful analogy is with the use of integral transforms when facing partial differential equations for functions of a continuous variable, such as time t . Examples of these include

$$\text{Fourier} \quad f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} d\omega \exp\{i\omega t\} \tilde{f}(\omega), \quad (\text{A.1})$$

$$\text{Laplace} \quad f(t) = \frac{1}{2\pi i} \int_{\gamma-iT}^{\gamma+iT} ds \exp\{st\} \tilde{f}(s), \quad (\text{A.2})$$

$$\text{Mellin} \quad f(t) = \frac{1}{2\pi i} \int_{\gamma-i\infty}^{\gamma+i\infty} dz t^{-z} \tilde{f}(z). \quad (\text{A.3})$$

(T)¹ In each case we replace a function of interest, in terms of one continuous variable, with new function in terms of a new continuous variable. Usually the original function is terms of a physical variable we can interpret easily (illustrated here by time t). The transformed function is expressed in terms of a variable may be much harder to interpret. Here the new variables are (angular) frequency ω , often with a natural interpretation, but s and z typically harder to interpret physically. Even if the variable is hard to interpret, the advantage is that if you choose the right transform then a solution should become apparent.

Generating functions are ubiquitous in physics. PARTITION FUNCTIONS in statistical mechanics are good examples. For instance, if the macrostates of the system are characterised by an energy and the number of particles in the system, then the partition function is defined as the sum over all possible states of the exponential of the energy and particle number, each term scaled by a parameter, the inverse temperature β and the chemical potential μ respectively. The result is a partition function, a function of the continuous variables — $Z(\beta, \mu) = \sum_i \exp\{-\beta(E_i - \mu N_i)\}$. We have switched from a description in terms of energy and particle number to a description in terms of the inverse temperature β and the chemical potential μ .

In quantum field theory, sums over different possible quantum field configurations ϕ (the different states) of $\exp\{iS/\hbar\}$ gives generating functions of the Green functions used in QFT which is a function of unphysical sources j — $Z[j] = \sum_\phi \exp\{-(i/\hbar)(S[\phi] - j\phi)\}$ for the action $S[\phi]$ of the theory.

Let us illustrate the use of generating functions for in the context of degree distributions $p(k)$. The aim is to package all the information recorded in the $p(k)$ in a GENERATING FUNCTION $G(z)$ by defining

$$G(z) = \sum_{k=0}^{\infty} z^k p(k) = p(0) + zp(1) + z^2 p(2) + \dots. \quad (\text{A.4})$$

Here the continuous variable z has no simple interpretation, and it is replacing the discrete variable k which is easily understood as the physical degree of nodes. The generating $G(z)$ still has all the information of $p(k)$ but it is wrapped up in a different mathematical form. Looking at (A.4) we see it is just a discrete version of an integral transform. **(T)²** There are several advantages to having a generating function

- We can invert by taking k -th derivative at $z = 0$

$$p(k) = \frac{1}{k!} \left. \frac{d^k G}{dz^k} \right|_{z=0}. \quad (\text{A.5})$$

Note that the analogy with the Mellin integral transform (A.3) shows another way to write the inverse

$$p(k) = \frac{1}{2\pi i} \oint dz z^{-k-1} G(z). \quad (\text{A.6})$$

¹**(T)**Relationship: $i\omega t \leftrightarrow st \leftrightarrow -z \ln(t)$.

²**(T)**Is this a Laplace transform as the integration must be lower bounded?

- The MOMENTS of distribution are easy to obtain

$$\langle k^n \rangle = \left(z \frac{d}{dz} \right)^k G(z, t) \Big|_{z=1} = \frac{d^k}{d(\ln(z))^k} G(z, t) \Big|_{z=1}. \quad (\text{A.7})$$

- the FACTORIAL MOMENTS of distribution are even easier to obtain

$$\langle k(k-1)\dots(k-n+1) \rangle = \left\langle \frac{\Gamma(k+1)}{\Gamma(k-n)} \right\rangle = \frac{d^n G}{dz^n} \Big|_{z=1}. \quad (\text{A.8})$$

We can use our (approximate) master equation for the BA model to illustrate another use of a generating function. Just as we can use integral transforms to rewrite PDE for functions of a continuous variable in a different, hopefully simpler form, we can use generating functions to replace a difference equation, such as (4.8.11), by a PDE for the generating function. Typically we are more experienced in finding solutions to a PDE.

So we start with the difference form of our Price master equation (4.8.11)

$$p_\infty(k^{(\text{in})}) = +\frac{m}{(m+\mu)}(k^{(\text{in})}+\mu-1)p_\infty(k^{(\text{in})}-1) - \frac{m}{(m+\mu)}(k^{(\text{in})}+\mu)p_\infty(k^{(\text{in})}) + \delta_{k^{(\text{in})},0} \quad \text{for } k^{(\text{in})} \geq 0. \quad (\text{A.9})$$

Noting that it is simpler to work with the variable $k = k^{(\text{in})} + \mu$ we multiply by z^k and then sum over k , using $G(z) = \sum_{k=\mu}^{\infty} z^k p_\infty(k-\mu)$. We find that³

$$G_\infty(z) = \frac{m}{(m+\mu)} z(z-1) \frac{\partial G_\infty(z)}{\partial z} + z^\mu. \quad (\text{A.10})$$

Now we have a PDE encoding the master equation, a much more familiar beast.

A.3 The Gamma and Beta functions

The GAMMA FUNCTION $\Gamma(x)$ plays a central role in many problems, and is particularly important when dealing with discrete quantities and DIFFERENCE EQUATIONS such as the master equation in the Price model in section 4.8.3.

The central property for the Gamma function, one that can be used to define the Gamma function, is that The central property for the GAMMA FUNCTION is that

$$\Gamma(z+1) = z\Gamma(z), \quad \Gamma(1) = 1, \quad (\text{A.1})$$

and this is for a complex number z . This means that for positive integer n we have⁴

$$\Gamma(n) \equiv (n-1)! \quad \text{for } n \in \mathbb{Z}^+. \quad (\text{A.2})$$

Put another way, the Γ function is the *unique* analytic continuation of the factorial function⁵ so it is the unique way to define a factorial for values other than positive integers.

³Try starting with the generating function form (A.10) and substitute in the definition of G to recover the original form of the master equation in terms of $p_\infty(k)$ (A.9).

⁴I prefer to reserve the factorial notation, $n!$, for positive integers n only.

⁵Technically you still need another condition, usually some condition on the behaviour of the function as $n \rightarrow \infty$. After all the function $\exp(2\pi iz)\Gamma(z)$ also has the same value as $\Gamma(z)$ at integer values of z but this first form oscillates and behaves badly at infinity.

A particularly useful example of Gamma function ratios comes as a solutions to equations of the following form. Consider the function $f(z)$ defined by

$$\frac{f(z)}{f(z-1)} = \frac{z+a}{z+b}. \quad (\text{A.3})$$

We find that this has the solution

$$f(z) = A \frac{\Gamma(z+1+a)}{\Gamma(z+1+b)} \quad (\text{A.4})$$

since

$$f(z) = A \frac{(z+a)\Gamma(z+a)}{(z+b)\Gamma(z+b)} = \frac{(z+a)}{(z+b)} f(z-1). \quad (\text{A.5})$$

Another property we will use is Stirling's approximation which in this language is the asymptotic expansion

$$\lim_{|z|\rightarrow\infty} \ln(\Gamma(z)) = z(\ln(z)-1) - \frac{1}{2}\ln(z/(2\pi)) + O(z^{-1}). \quad (\text{A.6})$$

In particular, from this you can show that

$$\lim_{|z|\rightarrow\infty} \frac{\Gamma(z+1+a)}{\Gamma(z+1+b)} = z^{a-b}. \quad (\text{A.7})$$

For numerical work, it is important to note that since $\Gamma(z)$ gets very large very quickly as $z \rightarrow \infty$ for real z (it rising like a factorial). This quickly overwhelms the numerical ability of any computer. So it is often not a good idea to work with the Γ function directly in code. Instead you should use the logarithm of the Gamma function which, as Stirling's formula shows (A.6), is much more reasonably behaved. For instance, to calculate a ratio of Γ functions numerically you should work numerically with the log of the ratio taking the exponential only at the end of the calculation. The ratio of gamma functions such as found in (A.4) rises much slower than factorial as (A.7) shows. So then what we use in numerical work is

$$\frac{\Gamma(z+1+a)}{\Gamma(z+1+b)} = \exp(\text{gammaln}(z+1+a) - \text{gammaln}(z+1+b)) \quad (\text{A.8})$$

where `gammaln` is the natural logarithm of the Γ function, i.e. $\ln(\Gamma)$. There is almost always a library routine for this, for instance for Python it can be found in `scipy`. These `gammaln` library functions give reasonable numerical values so the differences we need in (A.8) are also sensible. As long as $|a-b|$ is not too large then (A.7) shows us that the the final result for this ratio is not a factorially large number, so the final exponential on the right-hand side of (A.8) will work numerically too.

The BETA FUNCTION is (see 8.384 Gradshteyn and Ryzhik (1980))

$$B(x,y) = B(y,x) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}. \quad (\text{A.9})$$

This has an integral representation as

$$B(x,y) = \int_0^1 dt t^{x-1} (1-t)^{y-1} \quad (\text{A.10})$$

A most useful identity is (8.384.3) (Gradshteyn and Ryzhik, 1980) that

$$\sum_{n=0}^{\infty} B(x, y+n) = B(x-1, y), \quad (\text{A.11})$$

which can be derived from the integral representation (A.10) as follows

$$\sum_{n=0}^{\infty} B(x+n, y) = \int_0^1 dt \left(t^{x-1} \sum_{n=0}^{\infty} t^n \right) (1-t)^{y-1} \quad (\text{A.12})$$

$$= \int_0^1 dt \left(t^{x-1} \frac{1}{1-t} t^n \right) (1-t)^{y-1} \quad (\text{A.13})$$

$$= \int_0^1 dt t^{x-1} (1-t)^{y-2} = B(x, y-1) \quad (\text{A.14})$$

The version in (A.11) follows from the symmetry of the arguments of the Beta function.

A.4 Sets

A.4.1 Mathematical Sets

We will use set notation as a convenient shorthand to save writing out long sentences rather than exploiting its power. If you are uneasy with this, just write out the meaning of the terms in words alongside the formal notation until you get used to the notation. For a longer introduction to these topics I'd recommend Smith (1998).

A SET is a collection of *distinct* objects with no implied order. The objects in a set are called ELEMENTS or MEMBERS of a set. We use braces $\{\dots\}$ (curly brackets) to represent a set. For example suppose we have the integers 1, 2 and 3. Then these would be the three elements in our set and the set is denoted $\mathcal{S} = \{1, 2, 3\}$. Note that even though we know a natural order for the integers, the set does not care about any order so $\mathcal{S} = \{1, 2, 3\} = \{3, 1, 2\} = \dots$ all represent the same set.

Our example here has three elements, 1, 2 and 3. We show that these elements are in this set with the \in symbol, the SET INCLUSION symbol which means ‘is a member of’. Thus $1 \in \mathcal{S}$ means ‘1 is a member of the set \mathcal{S} ’. Equally \notin symbol means ‘is not a member of’ so in our example we have that $4 \notin \mathcal{S}$. The number of elements in a set is denoted as $|\mathcal{S}|$ so if $\mathcal{S} = \{1, 2, 3\}$ then $|\mathcal{S}| = 3$.

When we combine two sets this is called the UNION of the sets and is denoted by \cup . For instance, if we had $\mathcal{R} = \{A, B, C\}$ and $\mathcal{S} = \{A, C, D, E\}$ then the union of these two sets is denoted $\mathcal{R} \cup \mathcal{S}$ where $\mathcal{R} \cup \mathcal{S} = \{A, B, C, D, E\}$. Note how elements are never repeated in a set so in this union the elements A and C appear in both \mathcal{R} and \mathcal{S} but only once in their union $\mathcal{R} \cup \mathcal{S}$. Sometimes it is important to focus on only those elements that are common to two sets. For this we use set INTERSECTION operation denoted by \cap . For instance, $\mathcal{R} \cap \mathcal{S} = \{A, C\}$.

A SUBSET \mathcal{T} of a set \mathcal{S} is denoted $\mathcal{T} \subseteq \mathcal{S}$ and the subset is a set made only of elements of the first set \mathcal{T} . That is if $v \in \mathcal{T}$ then $v \in \mathcal{S}$. Note that valid subsets include the empty set \emptyset and the whole of the original set \mathcal{T} , so $\emptyset \subseteq \mathcal{T}$ and $\mathcal{T} \subseteq \mathcal{T}$. The symbol \subset is used to exclude the $\mathcal{T} \subseteq \mathcal{T}$ case.

A PROPER SUBSET \mathcal{T} of a set \mathcal{S} is denoted $\mathcal{T} \subset \mathcal{S}$. A proper subset \mathcal{T} is any subset of \mathcal{S} except for the whole set \mathcal{S} . Note that the empty subset is a proper subset⁶ $\emptyset \subset \mathcal{S}$.

One notation we will use from time to time indicates SET SUBTRACTION, more formally known as RELATIVE COMPLEMENT. That is where we define one set \mathcal{C} to be the set \mathcal{A} with any elements that are in \mathcal{B} removed. We will write this as⁷ $\mathcal{C} = \mathcal{A} \setminus \mathcal{B}$. For instance if $\mathcal{A} = \{1, 2, 3, 4\}$ and $\mathcal{B} = \{1, 3\}$ then $\mathcal{A} \setminus \mathcal{B} = \{2, 4\}$. Note that in all the examples we will use, \mathcal{B} is a subset

⁶This is different from group theory where a proper subgroup excludes both the whole original group *and* the smallest group, the trivial group of one element.

⁷The notation $\mathcal{C} = \mathcal{A} - \mathcal{B}$ is also common.

of \mathcal{A} but the definition given here is correct even if \mathcal{B} has elements not in \mathcal{A} as they do not change the result of the operation.

In set theory, a **COVER** of a set \mathcal{X} is a collection of subsets \mathcal{X}_n of \mathcal{X} whose union contains \mathcal{X} as a subset. One special type of cover is a **PARTITION**. A partition $\mathcal{P} = \{\mathcal{X}_n\}$ of a set \mathcal{X} is where we split up the set \mathcal{X} into a collection of smaller sets, $\{\mathcal{X}_n\}$, such that every element in the original set \mathcal{X} is in one and only one of the small subsets \mathcal{X}_n that make up the partition. That is we can define the partition \mathcal{P} of set \mathcal{X} as

$$\mathcal{P} = \{\mathcal{X}_n | n \in \{1, 2, \dots, P\}, \mathcal{X}_m \cap \mathcal{X}_n = \emptyset \text{ if } m \neq n, \cup_{m=1}^P \mathcal{X}_m = \mathcal{X}\}. \quad (\text{A.1})$$

There are occasions in mathematics when we need a collection of objects where each type of object can appear more than once but again there is no order in these elements and these are known as a **MULTISET** (also known as a **BAG**). For instance, a bag of coloured balls has no order in the balls but if balls of one colour appear more often it is important to record this in the mathematical representations, e.g. the probability of pulling out a ball of one colour depends on the number of balls of each colour in the bag.

It is also worth noting that when we have a set or multiset where there is a specified order to the elements, it is known as a **SEQUENCE**. We will use ordinary round brackets to denote a sequence. So if $\mathcal{Q} = (1, 2, 3)$ is a sequence then the order matters and so it is distinct from the sequence $\mathcal{Q}' = (1, 3, 2)$. With directed networks, we use a sequence of two elements, (u, v) , to represent a directed edge. This is one example of a path in a network where we can specify a path by the nodes it passes through in the order the nodes are reached as you follow the path.

At many points we will also need a **SEQUENCE** which is a set of points which come with a specific order. We will indicate this using parentheses (round brackets) $(1, 2, 3)$ where the order we write them in the notation, from left to write, defines the order required. That is in this example 1 comes before 2 and both come before 3. It is important as we may need to talk about a different quantity represented by the same numbers but in a different order, for example the sequence $(2, 1, 3)$. Now 2 comes before 1 and 3 is still after both 2 and 1. So these are different sequences. They are both made from the same *set* of elements $\{1, 2, 3\} = \{2, 1, 3\}$ but because we want the numbers in two different orders, we need to specify a sequence and use our (...) notation to indicate this.

Sometimes we need a briefer notation for sequences in which case we will use $(v_n)_{i=0}^L$ to indicate the sequence (v_0, v_1, \dots, v_L) . For example the traditional Fibonacci sequence is conveniently written as $\mathcal{F}_{\text{Fib}} = (f_n)_{i=0}^{\infty}$ along with the defining rule $f_{n+2} = f_{n+1} + f_n$ and initial conditions $f_1 = 0$ and $f_2 = 1$. Writing this out we see that $\mathcal{F}_{\text{Fib}} = (0, 1, 1, 2, 3, \dots)$ so unlike a set, a sequence can have repeated values.

A.4.2 Sets and Computer Languages

Many modern have some very useful ways of storing large amounts of information other than an array: lists or sets, ordered or not. These are known as **collections** in Java (for example see “Introduction to Programming Using Java” chapter 10 (Eck, 2007)), **containers** in the STL of C++ and simply data structures in Python. In fact in Python there is no array in the main language⁸ and the whole Python language is rooted in these more sophisticated ways of storing data.

The most common structures found in computer languages, such as a **Python list** or an **arrayC++!array** in C++, store data in a specific order, i.e. they are more like a sequence than a set. The computer language version of the mathematical set is less common and this is a more sophisticated type of data structure. In many cases the data structure is explicitly

⁸Arrays are part of packages added onto Python such as the **numpy** package.

called a **set**. The **set** in Java, **set** in Python and the STL **set** of C++ are the same as the mathematical set. Each of these computing structures will only ever hold one reference of an object. If you try to add object A to these computing **sets** and they already contain an A , then nothing changes. This is made clearer when you discover that these languages contain other types of data structure in which an element, technically a reference to a single value/object, can appear more than once in this type of collection. The list objects in Python, Java or C++ will do this job but are inefficient for this role in some ways. So you can often find special implementations of multisets such as the specific **multiset** in the STL of C++.

The behaviour of these computer **set** structures mimics what we actually want when defining a network e.g. a network includes a set of nodes where no node appears twice.

A sequence $\mathcal{Q} = (1, 2, 3)$ is a more natural structure for a computer and are found as arrays in most languages, such as C, but these tend to have a maximum length but this gives them a speed advantage. Data structures that are sequences but whose length can change as the programme runs are common in more recent languages, a **list** object in

A.4.3 Common probability distributions

The BINOMIAL DISTRIBUTION is denoted $B(N, p)$. We can visualise this as a situation where we have N trials, and on each trial with probability p we have a positive outcome and with probability $(1 - p)$ we have a negative outcome. The results of each trial is independent of any other trial but the probability p is the same for all trials. Then after N trials, we will find k positive results with probability $p(k)$ where

$$p(k) = \binom{N}{k} p^k (1 - p)^{N-k} \quad (\text{A.2})$$

We will use the POISSON DISTRIBUTION which is formally the large number of trials N , small probability p limit of the binomial distribution, where $\lambda = pN = \langle k \rangle$ is finite. That is we find the probability distribution is

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}. \quad (\text{A.3})$$

In many situations, N does not have to be very large before the Poisson distribution becomes a reasonable approximation to the binomial distribution so we will often use the Poisson even when the number of trials is finite.

Appendix B

Additional Material

In this chapter we will include some extra information on a variety of topics. This is largely incidental to the discussions in the main text but illustrates how some of the ideas can be generalised and developed. It also shows how some other work is related to what is discussed in the main text.

B.1 Disk Spaces and Formal Balls

For DISK SPACE see (Suzuki et al., 2019) and references therein. The idea is that we can define “disks” or more generally polyhedra in some d -dimensional metric space X (and more generally in a partial metric space) by specifying the perpendiculars to these boundaries. So working in a d -dimensional space we have a set of finite vectors in this space $\mathcal{W} = \{\mathbf{w}^{(i)}\}$. These are used to form a quasi-metric

$$d_{\mathcal{W}}(\mathbf{x}, \mathbf{y}) = \max_i \{\mathbf{w}^{(i)} \cdot (\mathbf{x} - \mathbf{y})\}. \quad (\text{B.1})$$

The quasi- here refers to the fact that this does not need to be symmetric so $d_{\mathcal{W}}(\mathbf{x}, \mathbf{y}) = d_{\mathcal{W}}(\mathbf{y}, \mathbf{x})$ need not be true. (T) Not clear to me why this was defined. This $d_{\mathcal{W}}$ is not used later but perhaps it is the example that enable you to define general disks as polytopes, i.e. closed regions bounded by polygons (end T) Q. What about

$$d_{p\mathcal{W}}(\mathbf{x}, \mathbf{y}) = \left(\sum_i [\mathbf{w}^{(i)} \cdot (\mathbf{x} - \mathbf{y})]_+^p \right)^{1/p}. \quad (\text{B.2})$$

where $[x]_+^p = x^p$ if $x > 0$ and is zero otherwise.

The shape of a DISK, often called a GENERALISED BALL for work in three or more dimensions, is fixed by the choice of $\{\mathbf{w}^{(i)}\}$. The position and size of individual disks are then specified by the centre $\mathbf{x} \in X$ and a size $r \in \mathbb{R}^+$ to scale the polytopes/disks. For a (quasi-)metric space (X, d) with a general distance measure d a closed DISK $D(\mathbf{x}, r)$ is defined as

$$D(\mathbf{x}, r) = \{\mathbf{y} \in X | d(\mathbf{y}, \mathbf{x}) < r\}. \quad (\text{B.3})$$

For instance, if $\mathbf{e}^{(i)}$ are basis vectors for the metric space X then we define $2d$ vectors $\mathbf{w}^{(2i-1)} = +\mathbf{e}^{(i)}$ and $\mathbf{w}^{(2i)} = -\mathbf{e}^{(i)}$ for $i = 1, 2, \dots, d$ and our disks are hypercubes in this space and their interior.

Note that experience with disks in *Euclidean* space suggests there is a natural partial order on any set of disks $\mathcal{D} = \{D_i = D(\mathbf{x}_i, r_i)\}$ through set inclusion, that is we can define

$$D_i \preceq D_j \Leftrightarrow D_i \subseteq D_j \quad (\text{B.4})$$

From Edalat and Heckmann (1998) (section 2.1) it appears that this condition $D_i \subset D_j \Rightarrow d(\mathbf{x}_j, \mathbf{x}_i) \leq r_j - r_i$ in general from the triangle inequality but *not* in general the reverse. Suppose $D_i \subset D_j$ then for all $\mathbf{z} \in D_i$ we have that $\mathbf{z} \in D_j$. We also have

$$d(\mathbf{x}_j, \mathbf{x}_i) + d(\mathbf{x}_i, \mathbf{z}) \leq d(\mathbf{x}_j, \mathbf{z}) \quad \forall \mathbf{z} \in D_i \subset D_j \quad (\text{B.5})$$

as the distance d is a (quasi-)metric and so obeys the triangle identity. In particular, we can always choose z_i to lie on the boundary of the disk D_i but it remains in disk D_j so that so then

$$d(\mathbf{x}_j, \mathbf{x}_i) + r_i \leq d(\mathbf{x}_j, \mathbf{z}) \leq r_j \quad d(\mathbf{x}_j, \mathbf{x}_i) \leq r_j - r_i \quad (\text{B.6})$$

In Suzuki et al. (2019) they note that in Euclidean space the distance between centres criteria does imply set inclusion $d(\mathbf{x}_j, \mathbf{x}_i) \leq r_j - r_i \Rightarrow D_i \subset D_j$. Let us see how that works to understand why it may not work in general. So suppose that $d(\mathbf{x}_j, \mathbf{x}_i) \leq r_j - r_i$. Let us choose a third point \mathbf{z} and then the triangle identity tells us that

$$d(\mathbf{x}_j, \mathbf{x}_i) + d(\mathbf{x}_i, \mathbf{z}) \leq d(\mathbf{x}_j, \mathbf{z}). \quad (\text{B.7})$$

In Euclidean space the largest value of the right hand side comes when we choose \mathbf{z} in line with the vector between the centres of the disks so that $\tilde{\mathbf{z}} = \mathbf{x}_i + \lambda(\mathbf{x}_j - \mathbf{x}_i)$ and

$$d(\mathbf{x}_j, \mathbf{x}_i) + d(\mathbf{x}_i, \tilde{\mathbf{z}}) = d(\mathbf{x}_j, \tilde{\mathbf{z}}). \quad (\text{B.8})$$

The largest value of $d(\mathbf{x}_i, \tilde{\mathbf{z}})$ comes when $\tilde{\mathbf{z}}$ is on the boundary opposite side of the disk from x_j where $d(\mathbf{x}_i, \tilde{\mathbf{z}}) = r_i$ giving us

$$d(\mathbf{x}_j, \mathbf{x}_i) + r_i = d(\mathbf{x}_j, \tilde{\mathbf{z}}). \quad (\text{B.9})$$

Since we assume $d(\mathbf{x}_j, \mathbf{x}_i) \leq r_j - r_i$ then we have $d(\mathbf{x}_j, \tilde{\mathbf{z}}) \leq r_j$ and hence we have set inclusion.

At the end of all this, it appears that we can define a partial order using disks in two ways. First using set inclusion (B.4). The second is by using the distance between centres

$$D_i \sqsubseteq D_j \Leftrightarrow d(\mathbf{x}_j, \mathbf{x}_i) \leq r_j - r_i. \quad (\text{B.10})$$

I think this is a *definition* of the \sqsubseteq symbol and a second partial ordering. In general these are not the same $D_i \sqsubseteq D_j \nLeftrightarrow D_i \prec D_j$ though this is true in Euclidean space. Not sure when it fails. Perhaps not in continuous spaces, see example in Edalat and Heckmann (1998) section 2.1.

The work of Edalat and Heckmann (1998) works in a metric space (X, d) and calls a FORMAL BALL as pair $(\mathbf{x}, r) \in \mathbf{BX}$ where $\mathbf{x} \in X$ and $r \in \mathbb{R}^+$ and adds an ordering \sqsubseteq from (??). Note there is *no* shape or definition to these balls, no use of the radius parameter to define a subset of points in X unlike the disk definition (C.1). Hence they are only “formally” a ball or disk.

Appendix C

Unused Material

C.1 Additional Network Material

(T) Not sure this paragraph is needed. (end T) There is a useful distinction made between certain types of subgraphs. A GRAPHLET (Pržulj, 2007) or INDUCED SUBGRAPH is the subgraph defined by a set of vertices along with *all* the edges between these vertices present in the graph. That is it is the largest possible given subgraph given the set of vertices. If a subgraph is not an induced subgraph then it is a PARTIAL SUBGRAPH. A subgraph is also sometimes called a MOTIF (Shen-Orr et al., 2002). These can be any type of subgraph but this term is usually used in a context where the number of such subgraphs found in a network is more abundant than expected when compared to a suitable random graph. A CLIQUE is a simple graph (or subgraph) where all the nodes are connected.

ASIDE Further Types of Subgraph

There are a few more terms associated with certain types of subgraph. (T) Not sure this is needed, even as a short paragraph in main text. (**end T**)

A CLIQUE is a simple graph (or subgraph) where all the nodes are connected. It is densest graph possible for the given set of vertices. One exercise is to try and find the CLIQUE COVER of a graph, that is to find a set of clique subgraphs, usually overlapping, such that every vertex and edge is at least one clique. Finding the smallest set of such cliques is a well studied problem in computer science. Cliques are also the simplest example of the type of dense regions that can play an important role in a network. Cliques, along with other dense subgraph such as CORES and COMMUNITIES, are examples of important mesoscopic structure that go beyond pairwise node organisation and they often represent emergent organisational patterns.

An INDUCED SUBGRAPH is the subgraph defined by a set of vertices along with *all* the edges between these vertices present in the graph. That is it is the largest possible given subgraph given the set of vertices. If a subgraph is not an induced subgraph then it is a PARTIAL SUBGRAPH.

A MOTIF (Shen-Orr et al., 2002) is just another name for one type of subgraph of interest in a larger graph. A motif is invariably small, and a typical example is that we are counting how many subgraphs are the same as (isomorphic to) the motif of interest. (T)^a We would typically then compare the number found against the same calculation performed on a null model. The original definition (Shen-Orr et al., 2002) stated a motif was always over represented though this does not always seem to be imposed when the term is used. Likewise, the original definition of motif (Shen-Orr et al., 2002) did not make it clear if the motifs were maximal subgraphs, i.e. induced subgraphs. So it maybe that the term is used with slightly different meanings. Note that when we find an example of the motif we are looking for in a larger graph, the vertices may have extra edges not included in the motif, that is it may be a partial subgraph. Thus we would find a motif of three vertices connected in a line in a triangle where three vertices are all connected directly to one another.

A GRAPHLET (Pržulj, 2007) is then the induced subgraph version of a motif. So if we are looking for a graphlet of three vertices connected in a line, a subgraph of a triangle of vertices would not count as being an example of this graphlet.

^a(T) From: Encyclopedia of Bioinformatics and Computational Biology, 2019. “Network motifs are partial subgraphs that are significantly over-represented in the data compared to a chosen random graph model that is assumed to fit well the data.”

C.2 Cube-Space DAG

Many other variations of the cube-space model exist. We can change the shape of the forward-connection kernel in many ways, such as demanding that $u_i < cv_i \forall i$ for some constant $c \geq 1$. One such model that has received some attention is where the forward-connection kernel is a cone centred along an axis parallel to a “time” coordinate $t = \sum_{i=1}^D x_i$. This is gives what is called a CONE-SPACE DAG!DIRECTED ACYCLIC GRAPH!CONE-SPACE DAG Bollobás and Brightwell (1991). It is important because this is the DAGs created by sprinkling points at random in Minkowski space-time and connecting them when they are causally connected (time-like separated) in the direction of causality, a model used in the causal set approach to quantum gravity ?. If we now compare to the original cube-space model, the forward connection kernels can both be seen as cones/pyramids, with an axis of symmetry along the time direction t . For simplicity, imagine we are in three dimensions then if we take a slice through the forward connection kernels along a plane perpendicular to the time axis we get a square for the cube-space case and a circle for the cone space. We can immediately see that we could chose this shape to be of any type, triangles, hexagons and so forth, and still get a consistent model. We can see they are consistent because we can view the condition in terms of these forard

cones/pyramids. Namely, a point u is connected to a point v only if the forward connection kernel of v is completely within the forward connection kernel of u . This is a form of the geometric set inclusion rule discussed in section 4.2.1. We can also change the angle these cones/pyramids subtend at the time axis (the examples above set this to 45°) and provided this angle is less than 90° we should get a consistent poset and DAG. It is unclear why we should want to chose any of these more general shapes, I have only seen examples of cube and cone space models. However, this picture links these spaces to the disk space models of section 4.2.1.

The Minkowski space-time DAG models are based on a distance measure that is invariant under a particular symmetry. In problems embedded in physical space and time, these symmetries makes the Minkowski space-time DAG models particularly relevant. However, in many data science problems we do not have these space-time symmetries in our data. The Minkowski space-time DAGs may still be useful tools as they the models encode the order inherent in the data if in one very particluar and interesting way. However, if this space-time symmetry is not present in the data, there is no good reason wy we should limit ourselves to the Minkowski space-time models.

One way to look for other models is to think about the way that the future set of any point in the Minkowski space-time models have the shape of a cone. So one way to generalise Minkowski space-time DAGs is to ask what other shapes we could use for the future set instead of a cone. As long as the future set defines a sense of order then we will obtain a DAG. We could choose many shapes but the most natural alternative is to use (hyper)-cubes to define the future set and these models are called CUBE-SPACE MODELS . These are defined in a D -dimensional real space so every node $v \in \mathcal{V}$ is associated with a D -dimensional vector of real coordinates v_i where $i = 1, 2, \dots, D$. We then define the edges to exist between two vertices $u, v \in \mathcal{V}$ if and only if the following CUBE-SPACE RULE holds

$$(u, v) \in \mathcal{E} \quad \text{iff} \quad u_i < v_i \quad \forall i = 1, 2, \dots, d. \quad (\text{C.1})$$

For example see Fig. C.1. In principle we can apply this hypercube definition of the future set of u for any set of points where each has D real values u_i associated with it. A wide number of data sets come with this vector representation of the data, for instance each point is a person and we know the age and height of that person to give us two “coordinates” for each person. Even if the data does not immediately have a vector representation, dat science often works by assigning a vector representation of each object. For instance if the nodes in our document represent documents then we can use the word frequencies in each document to produce vector representation for the document in a number of ways. The simplest would be to let u_i to represent the number of times document u contain word i . What is clear is that unlike the space-time example, each coordinate is almost always completely unrelated to another coordinate. That if v_1 is the age of the person i and v_2 is the height, there is no reason to consider linear mixtures of these two quantities. That is what the rotation symmetries that are part of the symmetries of Minkowski space-time are not relevant in most data science contexts and why cube-space, which has no implicit link between the different dimensions is probably more appropriate for generic data science contexts. Put another way, we could rescale any coordinate, e.g. measure the age of a person in days or years, their height in centimetres or inches, and it would make no difference to the connections made using the cube-space rule (C.1). Note that the sense of order comes from the inherent order of the real numbers used for each coordinate. So in our simple example used here the order is from shorter to taller people and from younger to older. It is far from obvious that this sense of order is relevant in our simple example. This reminds us that we need to decide from the context of the data if using a DAG representation with its inherent partial order is useful for our work.

In terms of a theoretical model, we will consider DAGs created from points laid down using a Poisson point process and connected using the cube-space rule (C.1). From now on we will use the term cube-space model to refer to this combination. The Poisson point process immediately gives us some important extra symmetries. In particular, the space looks identical from every possible point in the infinite version of this model, it is homogeneous and isotropic. Not, of course, for any one example, but on average, over many realisations of the model. These play a key role in the mathematical analysis of these models Bollobás and Brightwell (1991).

Figure C.1: An example of a cube-space DAG. Here five points are created where the coordinates (x_1, x_2) of each point \mathbf{x} are chosen uniformly at random to lie between 0 and 1. The range of locations of points is shown by the shaded background. These points are the five nodes of this network. In this case edges are chosen by defining the forward connection kernel to be a right-angled triangle where the right-angle is centred on each point and the short sides are equal length and parallel to axes. The forward connection kernel for point 1 is shown with a filled green triangle while for other points just the perimeter is shown by dotted lines. This results in the addition of five directed edges as shown by the solid black lines. DO PERMUTATIONS

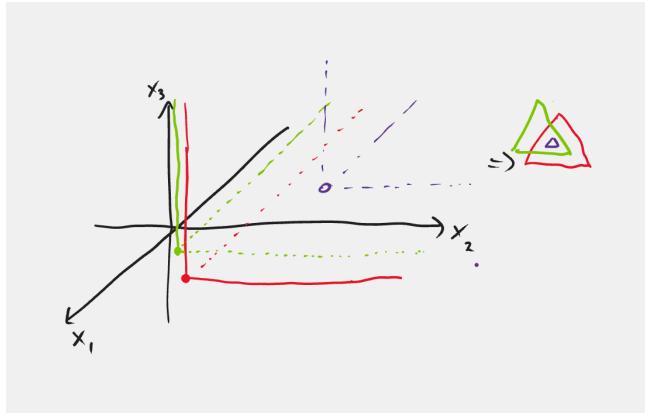


Figure C.2: This shows the forwards connection kernels in the cube-space DAG model which are cubes. The corner of each cube with the lowest (highest) coordinates is the coordinates linked to the associated node with that forward (backwards) connection kernel. There is an edge from one node to another if the forwards connection kernel of the source node completely contains that of the target node. Equally we can also take any plane orthogonal to the $(1, 1, 1)$ direction provided $t = \sum_i x_i$ for each node is smaller than the t value on this plane. The intersection of the forwards connection kernels gives equilateral triangles where the nodes can be represented at points at the centre of these triangles. Again there is an edge from one node to another if that node's triangle is completely contained within the triangle of another point.

There is another way to look at these cube-space models. Given that all that matters is the order of each coordinate, we can replace the real values v_i by the integer-valued rank. That is we could use $v_i^{(r)} \in \mathbb{Z}$ where there are $(v_i^{(r)} - 1)$ nodes u where $u_i < v_i$. More formally $v_i^{(r)} = |\mathcal{V}_i^-(v)| + 1$ where $\mathcal{V}_i^-(v) = \{u | u_i < v_i | u \in \mathcal{V}\}$. Now each coordinate is a permutation of 1 to N where we have N vertices in our model. It is then clear that we could choose to order our nodes using any one dimension, let's say the last coordinate $i = D$. So again, our nodes have a natural total order that is easy to exploit numerically. Given that each dimension is always naturally ordered (the real numbers assigned by the PPP as coordinates will never give a tie) it also implies that the non-trivial information in this model lies in the $(D - 1)$ other coordinates which are random permutations of one chosen dimension. More formally,

this view of the model as a set of $(D - 1)$ permutations chosen uniformly at random from the set of permutations of the numbers 1 to N has also been exploited mathematically. For example in Fig. C.1 we can choose to number our points so that they are vertical axis, x_2 BLAH BLAH

Again, the cube-space rule alone leads to a definition of a transitively complete DAG.

We can also think about how to impose a cutoff R in the size of the future set to produce models similar to RGG models. To do this we need some sense of distance. In the Minkowski space time models, the space-time comes equipped with the proper time providing a natural measure of distance and we used that to define a cutoff in (4.6.4). Here we have no sense of distance in our cube-space rule (C.1). However it turns out that when we impose the homogeneity and isotropic symmetries by using a constant average density of points in the PPP, this in fact gives a natural distance measure. The idea is that we can define the interval between two points u and v . The volume $\mu(u, v)$ of this interval $I(u, v) \subset \mathbb{R}^D$ is simply

$$\mu(u, v) = \int_{\mathbf{x} \in I(u, v)} d^D \mathbf{x} = \prod_{i=1}^D \int_{u_i}^{v_i} dx_i = \prod_{i=1}^D |v_i - u_i| \quad (\text{C.2})$$

where we have assumed the measure is simply $d^D \mathbf{x} = dx_1 dx_2 \dots dx_D$. We can always convert a volume into a length scale so we can define a natural length scale for the PPP with a cube-space rule as $d(u, v) = (\mu(u, v))^{1/D} = (\prod_{i=1}^D |v_i - u_i|)^{1/D}$ and, as with did with proper time, we can now define a pseudo-metric for cube-space as follows:-

$$d_{\text{Cu}}(x, y) = \begin{cases} \left(\prod_{i=1}^D (x_i - y_i)\right)^{1/D} & \text{if } x_i \geq y_i \forall i \\ \text{undefined} & \text{if } \exists j \text{ s.t. } x_j < y_j \end{cases} \quad (\text{C.3})$$

This has all the properties of what we called DAG metric in (3.3.5), that is

$$d_{\text{Cu}}(x, y) \geq 0 \text{ if } x \succeq y \quad \text{non-negativity}, \quad (\text{C.4a})$$

$$d_{\text{Cu}}(x, x) = 0, \quad \text{reflexivity}, \quad (\text{C.4b})$$

$$d_{\text{Cu}}(x, z) \geq d_{\text{Cu}}(x, y) + d_{\text{Cu}}(y, z) \text{ if } x \succeq y \succeq z \quad \text{reverse triangle identity}. \quad (\text{C.4c})$$

We could also use a closely related symmetric DAG metric definition where we replace $(x_i - y_i)$ in (C.3) by the modulus of this expression, $|x_i - y_i|$ so that the DAG metric then obeys the symmetry condition that $d(x, y) = d(y, x)$.

The only property which is not immediately obvious is the reverse triangle identity. We can show this by considering an interval from y to x split into two smaller intervals, one from y to a point w with coordinates $w_i = y_i + \lambda_i(x_i - y_i)$ where $0 \leq \lambda_i \leq 1$. Then we have that

$$d_{\text{Cu}}(x, w) + d_{\text{Cu}}(w, y) = \left(\prod_{i=1}^D (x_i - w_i) \right)^{1/D} + \left(\prod_{i=1}^D (w_i - y_i) \right)^{1/D} \quad (\text{C.5})$$

$$= \prod_{i=1}^D ((1 - \lambda_i)(x_i - y_i))^{1/D} + \prod_{i=1}^D (\lambda_i(x_i - y_i))^{1/D} \quad (\text{C.6})$$

$$= \left[\prod_{i=1}^D (1 - \lambda_i)^{1/D} + \prod_{i=1}^D (\lambda_i)^{1/D} \right] \prod_{i=1}^D ((x_i - y_i))^{1/D} \quad (\text{C.7})$$

One can check, by taking derivative with respect to λ_i , that the largest value for this expression is when $\lambda_i = 1/2$. **(T)¹**.

¹**(T)PROOF**

Since the cube-space rule does not depend on a measure of distance, we are free to work with a wide range of distance measures. We will just consider one better known alternative to (C.3) which is based on the L_p METRIC (also known as the MINKOWSKI DISTANCE though it has nothing to do with the Minkowski space-time where we use the proper time measure of distance). So we define The pseudo-distance function $d^{(Lp)}$ we will use we define as

$$d^{(Lp)}(x, y) = \begin{cases} [\sum_i |x_i - y_i|^p]^{1/p} & \text{if } x_i > y_i \forall i \text{ i.e. } x \succeq y, \\ \text{undefined} & \text{otherwise} \end{cases} \quad (\text{C.8})$$

where we always take the real root of $1/p$ operation. Again we can define a symmetrised version by replacing $(x_i - y_i)^p$ by the modulus $|x_i - y_i|^p$) and taking this value whenever $x \succeq y$ or $y \succeq x$.

The situation with the conditions of our DAG metric (3.3.5) is now more interesting. It is well known that the L_p measure of distance $\|\mathbf{x} - \mathbf{y}\|_p$ where

$$\|\mathbf{x} - \mathbf{y}\|_p = \left[\sum_i (x_i - y_i)^p \right]^{1/p} \quad \text{for } p \neq 0 \quad (\text{C.9})$$

is a metric for $p \geq 1$. That is it obeys the ordinary triangle identity (3.3.2). It is not too difficult to see the cube-space restriction in the $x \succeq y$ requirement does not alter this property except we now restrict the identity to be used only for

$$d^{(Lp)}(x, z) \leq d^{(Lp)}(x, y) + d^{(Lp)}(y, z) \text{ if } p \geq 1 \text{ and } x \succeq y \succeq z \text{ triangle identity.} \quad (\text{C.10})$$

What is more interesting is the case when $p < 1$. This is again tricky to show but we can do it for both $0 < p \leq 1$ and for $p < 0$. Normally these cases would not be considered as they are not metrics and the triangle identity (3.3.2) is not satisfied. However, we find that by combining the L_p definition of distance with the cube-space rule restriction in (C.8) we do get consistent results, with the triangle identity for $p \geq 1$ and the reverse triangle inequality for $p \leq 1$ (excluding $p = 0$)

$$d^{(Lp)}(x, z) \geq d^{(Lp)}(x, y) + d^{(Lp)}(y, z) \text{ if } p \leq 1, p \neq 0 \text{ and } x \succeq y \succeq z \text{ reverse triangle identity.} \quad (\text{C.11})$$

Again this is tricky to prove but we can do it. **(T)**².

So the model with the cube-space rule combined with the L_p metric is particularly interesting. We still get a DAG, the cube-space rule is sufficient to do this. However, the geometry of the underlying space of continuous variables is changing.

Why is the geometry of the underlying space of interest? Well one approach to any problem involving discrete data is to find an embedding in a continuous space where we can use the geometry of that space to achieve various goals in data science such as clustering. That is also an option when we have a network such as a DAG. We might try to understand the properties of that DAG by embedding in a continuous space and then using our understanding of that continuous space to solve network problems such as community detection (i.e. node clustering) or centrality measures.

One central issue in any space, be it a discrete one such as a DAG, or a continuous space such as a ... **(T)**³

(T)* CONTINUE FROM HERE *** (end T)**

²**(T)PROOF**

³**(T)GEODESIC**

C.3 Box Space

This is a summary of some of the key points of Bollobás and Brightwell (1991) who define box space.

The POWER SET $P(X)$ of some set X is the set of all possible subsets of X including X and the empty set \emptyset .

A σ -ALGEBRA Σ is a set of subsets of X (i.e. a subset of the power set $P(X)$) that obeys the following three properties⁴

1. Σ is closed under complementation:

$$A \in \Sigma \Rightarrow A^c = X \setminus A \in \Sigma. \quad (\text{C.1})$$

2. Σ is closed under countable unions:

$$A = \bigcup_i A_i \in \Sigma \quad \forall A_i \in \Sigma. \quad (\text{C.2})$$

3. Σ is closed under countable intersections:

$$A = \bigcap_i A_i \in \Sigma \quad \forall A_i \in \Sigma. \quad (\text{C.3})$$

It follows from this that

$$\emptyset \in \Sigma, \quad (\text{C.4})$$

$$A = \bigcap_i A_i \in \Sigma \quad \forall A_i \in \Sigma. \quad (\text{C.5})$$

A MEASURE μ is a map from a σ -algebra over a set X to the reals (or more generally a field as used in (Bollobás and Brightwell, 1991)), $\mu : \Sigma(X) \rightarrow \mathbb{R}$ that obeys the following

1. Non-negativity:

$$\mu(E) \geq 0 \quad \forall E \in \Sigma. \quad (\text{C.6})$$

2. Null empty set:

$$\mu(\emptyset) = 0. \quad (\text{C.7})$$

3. Countable additivity on disjoint subsets:

$$\mu\left(\bigcup_i A_i\right) = \sum_i \mu(A_i) \quad \text{where} \quad A_i \in \Sigma, \quad A_i \cap A_j = \emptyset \text{ if } i \neq j. \quad (\text{C.8})$$

In principle a measure can represent any sort of measurement made on a set e.g. lengths, areas, volumes. However I think if you have a d' -dimensional measure in a d -dimensional space where $d > d'$ then we can always project down onto a d' -dimensional space without losing any information. For instance if the measure is the sum of coordinates $\mu(\mathbf{x}) = x'_0 = (d)^{-1} \sum_{i=1}^d x_i$ then we just change variables to x'_0 and $x'_i = x_i - x'_0$ ($i = 1, 2, \dots, (d-1)$) and we can then ignore the x'_i for $i > 0$ as they have no effect, i.e. we are essentially in a one-dimensional space. Late

⁴An alternative definition I found was as follows. (i) X is in Σ , and X is considered to be the universal set in the following context. (ii) Σ is closed under complementation. (iii) Σ is closed under countable unions.

in Bollobás and Brightwell (1991) (see page 66) it appears that we are assuming that $d\mu(x)$ is the effective volume element of the space. (T)⁵

We can see that we can break any non-empty element of the σ -algebra down into constituents parts, a set of disjoint elements. If these that do not contain any subsets of the σ -algebra and these can be used to construct any other set in the σ -algebra. Suppose that B is a proper subset of A , $B \subset A$ and $B \neq \emptyset$. Further suppose both are in the σ -algebra. Then it follows that $A^c \in \Sigma$, so $C = A^c \cap B \in \Sigma$ with $B \cap C = \emptyset$. Thus $A = B \cup C$. That is, if any element of the σ -algebra has one smaller (non-empty) subset that is also in the σ -algebra then we can split the larger element into these two parts. So we can think of every element of the σ -algebra as the union of distinct ‘constituent partition’ (my language for a set of ‘basis’ sets) from which we can build the whole σ -algebra. In principle, if the σ -algebra has an infinite number of elements, this process might never stop and these building blocks would be the infinitesimal measures $\prod_i dx_i f(\mathbf{x})$ that we see in integration. Put another way, if $\{A_i\}$ is a countable partition of X then the collection of all unions of sets in the partition (including the empty set) is a σ -algebra. The converse is also (?) true. If we are given a set of subsets $\{B_j\}$ whose union is X and break these subsets down into the partition with the smallest number of sets that covers all of those subsets, i.e. partition $\{A_i\}$ s.t. every B is a union of some of the A_i in the partition, then this partition generates everything to do with the σ -algebra.

It then that because B and C are disjoint we have that $\mu(A) = \mu(B) + \mu(C)$ so $\mu(A) \geq \mu(B)$. In general the measure obeys

$$\mu(A) \geq \mu(B) \text{ if } B \subseteq A, \quad A, B \in \Sigma \quad (\text{C.9})$$

as shown in Fig. C.3.

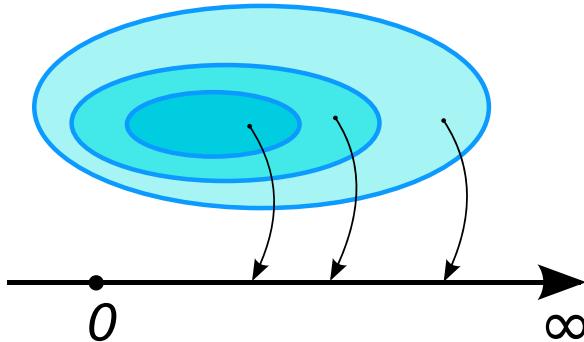


Figure C.3: An illustration of a measure. [From Wikipedia].

Note the countability/additivity criteria is telling us the following. Suppose we break the elements of the σ -algebra, down into the elements of the ‘constituent partition’ $\{A_i\}$ (the smallest partition whose unions cover every element of the σ -algebra). So suppose $A = \bigcup_i A_i$ where $A_i \cap A_j = \emptyset$ if $i \neq j$ and if $B \subset A_i$ with both in Σ then either $B = \emptyset$ or $B = A_i$. Then we know $\mu(A) = \sum_i \mu(A_i)$. So whenever we have an example where we have some natural building blocks, we must be able to do this simple sum. For instance, on a square lattice, we could say the measure of each point is 1.0 (the area of the Voronoi cell centred on the lattice point). Then we must say that the area of any collection of points is the sum of areas, the number of points. We could not state that this was the number of points to some power other than one as we need additivity, linearity.

A MEASURE SPACE is the triple (X, Σ, μ) .

⁵(T)For instance, if $\mu(\mathbf{x}) = x'_0$ then $d\mu(\mathbf{x}) = dx'_0$. Had we tried to work in the full $d^d \mathbf{x}$ space, what would happen?

A PARTIALLY ORDERED MEASURE SPACE (X, Σ, μ, \prec) is a measure space (X, Σ, μ) with a partial order so (X, \prec) is a poset (see elsewhere for definitions). An interval $I(x, y)$ (denoted $\langle x, y \rangle$ in Bollobás and Brightwell (1991)) is the closed set

$$I(x, y) = \{z | x \preceq z \preceq y\} \in \Sigma \quad x \prec y, \mu(x, y) > 0. \quad (\text{C.10})$$

Consider two partially ordered measure spaces (X, Σ, μ, \prec) and $(X', \Sigma', \mu', \prec')$ linked by a bijection λ where $x' = \lambda(x) \forall x \in X, x' \in X'$. These are said to be ORDER ISOMORPHIC if (i) $x < y$ iff $x' = \lambda(x) <' \lambda(y) = y'$ and (ii) $\lambda(\Sigma) = \Sigma'$. I assume the latter condition means every element in the σ -algebra is mapped appropriately to respect the operations of the two σ -algebras. If we also have (iii) that there is a SCALE-FACTOR α such that $\mu(A') = \mu(\lambda(A)) = \alpha\mu(A) \forall A \in \Sigma$ then we say that the two spaces are SCALE-ISOMORPHIC. Finally we say a partially ordered measure space is HOMOGENEOUS if every pair of intervals in X (**T**)⁶ are scale-isomorphic.

Note that if the bijection is a simple scaling off coordinates by constant λ the the scale factor is $\alpha = (\lambda)^{-d}$ so this already gives us the dimension d of the space. It is not clear to me if the scale-isomorphic condition guarantees

So in a HOMOGENEOUS PARTIALLY ORDERED MEASURE SPACE, an HPO-SPACE in Bollobás and Brightwell (1991), we can rescale it so that any interval is equivalent to the interval between the origin $\mathbf{0} = (0, 0, \dots)$ and the one-vector $\mathbf{1} = (1, 1, \dots)$. So a BOX SPACE is a homogeneous partially ordered measure space defined on the interval $I(\mathbf{0}, \mathbf{1})$.

Bollobás and Brightwell (1991) show a number of theorems. The first theorem states that

Theorem C.1. *For every box space (X, Σ, μ, \prec) the the DIMENSION of the box space is a constant $d \in [1, \infty]$ such that $V_n = n^{-d}$ for every $n \in \mathbb{N}$ where*

$$V_n = \sup_{\mathcal{P}_n} \left[\min_{i=0, \dots, n-1} \mu(\mathbf{x}_i, \mathbf{x}_{i+1}) \right], \quad \mathcal{P}_n = \{\mathbf{x}_i | i = 0, \dots, n; \mathbf{x}_0 = \mathbf{0}, \mathbf{x}_n = \mathbf{1}; x_i \prec x_{i+1}\}. \quad (\text{C.11})$$

That is for any path of length n from the source to sink of the interval $I(\mathbf{0}, \mathbf{1})$, we take the smallest value of the measure on any one of the n segments in the path and look for the maximum value that can take for any path. What it means is that we will find the path where the measure takes the same value on every segment provided increasing the value on one segment by moving one of the end points will always then reduce the value on another segment. So for instance, a path (the path?) that gives the value in cube-space is where $\mathbf{x}_i = (i/n, i/n, \dots, i/n)$.

This example makes it clear that this is a property of the dimension of *measure* and not the space as we could define the measure to be some sort of hyper-volume, $\mu(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^d (y_i - x_i)$ which is d dimensional, or a length scale such as $\mu(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^d (y_i - x_i)$ which is one dimensional, remembering that we always define by rescaling properties $\mu(\mathbf{0}, \mathbf{1}) = 1$. Note this appears to be easy to understand if we have translation invariance $\mu(\mathbf{x}, \mathbf{y}) = \mu(\mathbf{x} + \mathbf{z}, \mathbf{y} + \mathbf{z})$ on top of the scaling property, as we do in cube-space as the interval represented by each segment can be rescaled to match the original $I(\mathbf{0}, \mathbf{1})$. The proof in Bollobás and Brightwell (1991) does not seem to use this translation property.

The second theorem of Bollobás and Brightwell (1991) states that

Theorem C.2. *For a box space (X, Σ, μ, \prec) of dimension d then so any $0 < U < 1$ we have that $R(U) = (1 - U^{1/d})^d$ where*

$$R(U) = \sup_{\mathbf{x} \in X} \{\mu(\mathbf{x}, \mathbf{1}) | \mu(\mathbf{0}, \mathbf{x}) \geq U\}. \quad (\text{C.12})$$

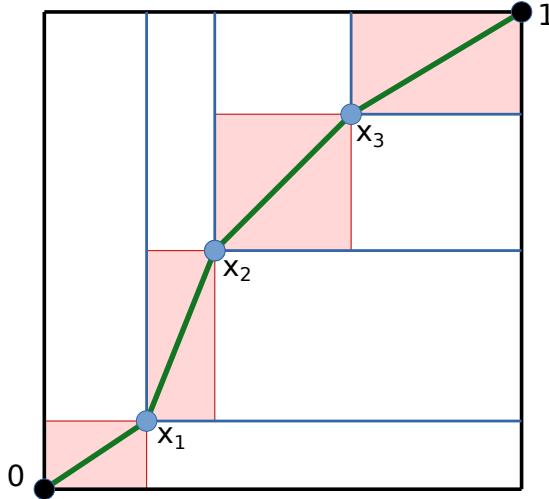


Figure C.4: An illustration of theorem 1 of Bollobás and Brightwell (1991). We applied the measure to each of the three segments in this path \mathcal{P}_3 , e.g. the area of the interval or the circumference, and we take the smallest value, say $\nu(\mathcal{P}_3)$. We then look at all paths of length 3 and take the largest value of $\nu(\mathcal{P}_3)$, this is V_3 . When we have scale and translation invariance, it is pretty obvious the value of V_n comes from the case where all intervals are the same, just rescaled versions of the interval $I(\mathbf{0}, \mathbf{1})$.

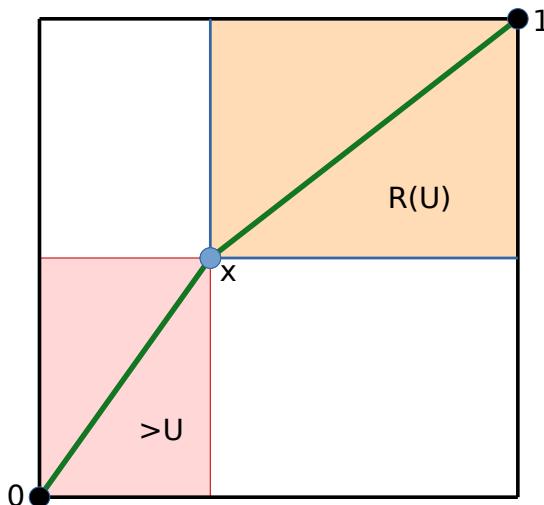


Figure C.5: An illustration of theorem 2 of Bollobás and Brightwell (1991). For some given U , the interval from the sink to the point \mathbf{x} has a value of measure $\mu(\mathbf{0}, \mathbf{x}) \geq U$. Then the largest possible value the measure on the upper interval can take is $R(U) = (1 - U^{1/d})^d$.

We can see how for rational values of U this follows from scaling and theorem 1.
A most interesting part of this is the corollary

Corollary C.3. *For $\mathbf{x} \prec \mathbf{y} \prec \mathbf{z}$ in a box space (X, Σ, μ, \prec) of dimension d then*

$$[\mu(\mathbf{x}, \mathbf{y})]^{1/d} + [\mu(\mathbf{y}, \mathbf{z})]^{1/d} \leq [\mu(\mathbf{x}, \mathbf{z})]^{1/d} \quad (\text{C.13})$$

This is just a reverse triangle identity for a length scale $[\mu(\mathbf{x}, \mathbf{y})]^{1/d}$ of an interval $I(\mathbf{x}, \mathbf{y})$. It follows from the theorem as we can set $\mathbf{x} = \mathbf{0}$, $\mathbf{z} = \mathbf{1}$. Then for our fixed chosen \mathbf{y} we choose U such that $\mu(\mathbf{0}, \mathbf{y}) = U$. Then by Theorem C.5 we have that $\mu(\mathbf{y}, \mathbf{1}) \leq R(U) = (1 - U^{1/d})^d$. It follows that $[\mu(\mathbf{0}, \mathbf{y})]^{1/d} = U^{1/d}$ and $[\mu(\mathbf{y}, \mathbf{1})]^{1/d} \leq [R(U)]^{1/d} = (1 - U^{1/d})^d$. Then

⁶(T) Surely in Σ ?

$[\mu(\mathbf{0}, \mathbf{y})]^{1/d} + [\mu(\mathbf{y}, \mathbf{1})]^{1/d} \leq U^{1/d} + (1 - U^{1/d}) = 1 = [\mu(\mathbf{0}, \mathbf{1})]^{1/d}$. I think rescaling then allow us to say it is true for any \mathbf{x} and \mathbf{z} .

In fact we can show this reverse triangle identity (C.13) in a trivial manner. We know that $\mu(\mathbf{x}, \mathbf{y}) + \mu(\mathbf{y}, \mathbf{z}) \leq \mu(\mathbf{x}, \mathbf{z})$. This follows because the interval $I(\mathbf{x}, \mathbf{z})$ contains \mathbf{y} and so all points on paths from \mathbf{x} to \mathbf{z} via \mathbf{y} are included in $I(\mathbf{x}, \mathbf{z})$. So the volume of the interval $I(\mathbf{x}, \mathbf{z})$ must be at least the volume of the two smaller intervals $I(\mathbf{x}, \mathbf{y})$ and $I(\mathbf{y}, \mathbf{z})$. However we also have the inequality that $x^q + y^q < (x + y)^q$ for $x, y > 0$ and $0 < q < 1$. I cannot find a reference or name for this inequality but we can prove it by choosing w.l.o.g. $y > x$ and considering $f(z) = (1 + z)^q - 1 - z^q$ where $0 < z = x/y < 1$ and $z = x/y$. Differentiating $f(z)$ shows it is monotonic in this region and $f(z = 0) = 0$. Returning to the intervals, we see that

$$[\mu(\mathbf{x}, \mathbf{y})]^q + [\mu(\mathbf{y}, \mathbf{z})]^q \leq [\mu(\mathbf{x}, \mathbf{y}) + \mu(\mathbf{y}, \mathbf{z})]^q \leq [\mu(\mathbf{x}, \mathbf{z})]^q, \quad \text{for } 0 < q < 1 \quad (\text{C.14})$$

Thus we recover (C.13) for $q = 1/d$. However, this approach shows that $q = 1/d$ is not special in terms of this identity. To get a quantity that has the correct scaling as a length, we need $q = 1/d$, but we do not need it for (C.13).

Thus it appears from (C.13) and checking the other properties of a space-time type distance that box spaces, including cube-space, already has a reverse metric $d_{(B)}(\mathbf{x}, \mathbf{y})$ where this BOX SPACE METRIC is defined as

$$d_{(B)}(\mathbf{x}, \mathbf{y}) = d_{(B)}(\mathbf{y}, \mathbf{x}) = (\mu(\mathbf{x}, \mathbf{y}))^{1/d} \quad \text{where } \mathbf{x} \prec \mathbf{y}. \quad (\text{C.15})$$

That is we have from Definition ?? (see also Partial Order Metric ??) that when $x \preceq y \preceq z$ then

$$d_{(B)}(x, y) \geq 0, \quad \text{non-negativity,} \quad (\text{C.16a})$$

$$d_{(B)}(x, y) = d_{(B)}(y, x), \quad \text{symmetry,} \quad (\text{C.16b})$$

$$d_{(B)}(x, x) = 0, \quad \text{reflexivity,} \quad (\text{C.16c})$$

$$d_{(B)}(z, x) \geq d_{(B)}(z, y) + d_{(B)}(y, x), \quad \text{reverse triangle inequality} \quad (\text{C.16d})$$

Lemma C.4. *Conjectured Corollary. A point y lies on a geodesic from x to z when in the reverse triangle inequality (C.16d) we have equality.*

Lemma C.5. *Conjectured Corollary. It seems that we only get equality in (C.16d) when y lies on the geodesic from x to z .*

One of the interesting observations for the cube spaces is the close similarity with distances L_p . Suppose we add the L_p (reverse-)metric to the the cube-space. However nothing to stop us measuring distance in another way. In that case we have the reverse triangle inequality

$$\left[\sum_{i=1}^d |y_i - x_i|^p \right]^{1/p} + \left[\sum_{i=1}^d |z_i - y_i|^p \right]^{1/p} \leq \left[\sum_{i=1}^d |z_i - x_i|^p \right]^{1/p}, \quad \text{for } 0 < p < 1 \quad (\text{C.17})$$

Note again that any value of p can be chosen. If we choose $p = d$ then the arguments of the $[...]^{1/p}$ scale like volumes but there is no reason to do that. The comparison of the measure based metric and the imposed L_p metric is that we are comparing

$$\mu(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^d |y_i - x_i| \quad \text{vs.} \quad \sum_{i=1}^d |y_i - x_i|^d \quad (\text{C.18})$$

Put another way, the Box space reverse metric $d_{(B)}(x, y)$ becomes the cube-space metric $d_{Cu}(x, y)$ where

$$d_{Cu}(x, y) = \left(\prod_{i=1}^d |y_i - x_i| \right)^{1/d} \quad \text{vs.} \quad d_p(x, y) = \left(\sum_{i=1}^d |y_i - x_i|^p \right)^{1/p} \quad (\text{C.19})$$

C.4 Extended Family of Metrics

Let us illustrate the idea by using three dimensions. We will consider the distance between the origin $\mathbf{0} = (0, 0, 0)$ and the point $\mathbf{p} = (x, y, z)$. For our context of DAGs and cube-space rules⁷, we have that $x, y, z \geq 0$. We could also replace x by $|x|$ and so forth in what follows if we wanted to avoid the cube-space direction rule.

It is useful to define the cuboid C formed by corners of the eight points: $\mathbf{0} = (0, 0, 0)$, $(x, 0, 0)$, $(0, y, 0)$, $(0, 0, z)$, $(x, y, 0)$, $(0, y, z)$, $(x, 0, z)$ and $\mathbf{p} = (x, y, z)$. This encloses the volume which forms the interval (Alexandrov set) defined by the points at the opposite corners $\mathbf{0} = (0, 0, 0)$ and $\mathbf{p} = (x, y, z)$. That in the cube-space rules this is the volume containing all the points which lie on a directed path from $\mathbf{0} = (0, 0, 0)$ to $\mathbf{p} = (x, y, z)$.

First principle is that we wish the distance to have units of the coordinates. That is if we scale the coordinates by a constant λ then the distance measure does the same, so that

$$d(\lambda\mathbf{0}, \lambda\mathbf{p}) = \lambda d(\mathbf{0}, \mathbf{p}) \quad (\text{C.1})$$

where $\lambda\mathbf{p} = (\lambda x, \lambda y, \lambda z)$. Note we need to be able to define scalar multiplication of our vectors so we are again assuming natural vector space properties for our vectors and their coordinates.

Another key principle in a data science context might be that we demand any distance d is invariant under permutations of the coordinates. In our case $d(\mathbf{0}, \mathbf{p}) = d(\mathbf{0}, (x, y, z)) = d(\mathbf{0}, (y, x, z))$ etc. This is not as powerful of full rotation symmetry of which this permutation symmetry is a subset but it does give us a natural way to produce a family of distance measures.

It may well be that I am assuming we have translation invariance too so that

$$d(\mathbf{q}, \mathbf{p} + \mathbf{q}) = d(\mathbf{0}, \mathbf{p}) \quad (\text{C.2})$$

where we have natural definition of the addition on vectors, i.e. coordinate by coordinate.

We may also want a flat space where

$$d(\mathbf{0}, \mathbf{p}) = d(\mathbf{0}, \lambda\mathbf{p}) + d(\lambda\mathbf{p}, \mathbf{p}), \quad 0 \leq \lambda \leq 1. \quad (\text{C.3})$$

Essentially we are saying that the points $\lambda\mathbf{p}$ lie on a “straight” line from origin to \mathbf{p} . This is one definition of a geodesic.

One property we may want to check is the triangle inequality. So we define

$$\Delta = d(\mathbf{0}, \mathbf{q}) + d(\mathbf{q}, \mathbf{p}) - d(\mathbf{0}, \mathbf{p}) \quad \forall \quad 0 \leq q_i \leq p_i. \quad (\text{C.4})$$

This value Δ is called the GROMOV PRODUCT for instance see Babul et al. (2022). If $\Delta \geq 0$ we have points that satisfy the what is called the triangle inequality. If $\Delta \leq 0$ we have points that satisfy what I call the reverse triangle inequality. The set of points \mathbf{q} which have zero Δ is another way to define a geodesic between two points. This means the geodesic is a set of points that saturate the triangle inequality. Numerical studies of these values have been done (`distancevariants.py`) with one thousand points and \mathbf{p} is the three-dimensional vector with coordinates all 1.0.

At first order in the coordinates, we have only one option with this scaling property and permutation symmetry, that is

$$d(\mathbf{0}, \mathbf{p}) = x + y + z \quad (\text{C.5})$$

This is the Manhattan L_p metric for $p = 1$. It is a metric but it is also a reverse metric, it on the boundary between both of them. That is the triangle identity is always satisfied which is the

⁷That is we have an order, a poset structure. This is where we only have a connection, and therefore a valid distance, from source point s to target point t if every coordinate of t is greater than the corresponding coordinate of s . That is s is connected to t if $s_i < t_i$ for all i .

boundary case of the inequality. The straight line property (C.1) is trivially always true so the triangle identity is the boundary value, $\Delta = 0$ which this value of Δ is again another definition of a geodesic between two points. In terms of Box spaces, what we are doing is projecting our three-dimensional Cube box space onto a one-dimensional space with coordinates $X = x + y + z$ which is a simply a Cube box space with (volume) measure $\mu(\mathbf{0}, \mathbf{p}) = (x + y + z)$. We need to consider the idea that we can use the length of the edges, which is four times (C.5), of the cuboid as a measure on the three-dimensional space. That is we have to project this into a 1D space in order for this to be a suitable measure we can use for a Box space.

At second order in the coordinates, we have two options with this scaling property and permutation symmetry, that is

$$d(\mathbf{0}, \mathbf{p}) = (x^2 + y^2 + z^2)^{-1/2} \quad (\text{C.6})$$

$$d(\mathbf{0}, \mathbf{p}) = (xy + yz + zx)^{-1/2} \quad (\text{C.7})$$

The straight line property (C.1) is trivially always true for both so the “straight line”.

The first (C.6) is the Euclidean metric, the L_p metric for $p = 2$. Can it be used to define a measure in this space and hence a Box space? The fact it is a metric and not a reverse metric suggests in 2D at least, this cannot be used to be a measure and we cannot reach a Box space from here. A numerical study confirms that this satisfies the triangle inequality.

The second one using the total surface area of the cuboid C . A number of questions exist for this second case(C.7). First does it obey the triangle or reverse triangle identity or neither, so is this a metric or is it a reverse metric. A numerical study confirms that this satisfies the reverse triangle inequality. Yet to find a proof in general. Secondly, is $(xy + yz + zx)$ an appropriate measure on the space? The answer to this second question is no. Think of putting a point in the middle of the cuboid C . We can use this to partition the cuboid into eight non-overlapping cuboids. However the sum of the surface areas of these small cuboids will be twice the surface area of the large cuboid, that is because all the internal surfaces of the small cuboids do not contribute to the large cuboid surface area but they did to the small ones So we cannot use this to define a measure in the three-dimensional space and hence this cannot be used to define a Box space. However cannot the same argument be used for (C.5) which is related to the length of the edges of the cuboid C . We can only form a Box space by projecting down onto a two-dimensional space. Can we do that consistently?

At third order in the coordinates, we have three options with this scaling property and permutation symmetry, that is

$$d(\mathbf{0}, \mathbf{p}) = (x^3 + y^3 + z^3)^{-1/3} \quad (\text{C.8})$$

$$d(\mathbf{0}, \mathbf{p}) = (xy^2 + xz^2 + yz^2 + yx^2 + zx^2 + zy^2)^{-1/3} \quad (\text{C.9})$$

$$d(\mathbf{0}, \mathbf{p}) = (xyz)^{-1/3} \quad (\text{C.10})$$

The first (C.8) is L_p for $p = 3$ so this is a metric. Again, the fact it is a metric and not a reverse metric suggests in 3D at least, this cannot be used to be a measure and we cannot reach a Box space from here. Again the numerical study confirms what we know that this satisfies the triangle inequality.

The third (C.10) is the standard cube-space reverse metric as it is based on the volume measure $\mu(\mathbf{0}, \mathbf{p}) = (xyz)$. So we have all the usual Box space properties. The numerical study shows what we know from Bollobás and Brightwell (1991) that this always satisfies the reverse triangle inequality.

The unknown case is the second one C.9. Same questions as above, what is the triangle inequality relation, can we use this as a measure to form a box space in 3D or can we find a projection to a lower dimensional Box space. The numerical study shows that Δ for this

distance function can take either value. That suggests this might not give a useful distance function or metric.

C.5 Disk Spaces and Box Spaces

(T) Disk spaces are covered in set inclusion. (end T)

It is clear, as noted in Suzuki et al. (2019) in section 3.1, that Cone space Co_D of Bollobás and Brightwell (1991), i.e. D -dimensional Minkowski space-time is an example of a disk space. Look at a space-like hypersurface, say at constant time later than the time of all points (events). WLOG choose this surface to be at $t = 0$, then the light cones of an event at (t, \mathbf{x}) (t is negative in this definition) intersect the time-like surface to define $(d - 1)$ -dimensional spheres of radius $r = -ct$ (choosing units where speed of light $c = 1$) centred at \mathbf{x} and these are the disks or generalised balls in this model. The hypersurface is a Euclidean $(D - 1)$ -dimensional space and thus set inclusion of these light cones and the Euclidean distance between radii give the same poset. However, note that we have to impose a distance and metric on the cone space or at least its spatial hypersurface which, while natural, is not needed. In some sense the set inclusion definition is all we need.

The cube-space Cu_D of Bollobás and Brightwell (1991) needs a different perspective. Now it appears that we project the forward connection kernels, hypercubes in this model onto a (flat?) plane which cuts all the axes at positive values. That is we project onto a plane perpendicular to any vector \mathbf{p} where $p_i > 0$ for all $i = 1, 2, \dots, D$. This defines triangles in $D = 3$ and in general simplices with D vertices on the $(D - 1)$ dimensional hypersurface. This is because each line from a point which is parallel to an axis defines one edge of the forward connection kernel and each of these edges gives a single point on the $(D - 1)$ dimensional hypersurface. This is the content of theorem 1 in Suzuki et al. (2019) (see appendix for detailed proof and Fig 4a for $D = 3$ example) where cube-space is called ORDER EMBEDDING, (Vendrov et al., 2016, Suzuki et al., 2019).

In general we never needed to limit ourselves to cones and hypercubes for the shape of the forward connection kernel (light cone) in these models. We could easily define the forward connection kernel using a constant set of $n \geq D$ or more vectors $\mathbf{w}^{(i)}$, and these would be the edges of the forward connection kernels from any point, i.e. $\mathbf{x} + \lambda\mathbf{w}^{(i)}$ for $\lambda > 0$. The limit of $n \rightarrow \infty$ presumably interpolates between cube $n = D$ and cone space $n = \infty$.

Some comments. There is no distance/metric in the cube-space so we cannot use the distance measure on the disk space project, only set inclusion. We could change the angle subtended by the boundaries, i.e. change $c = \tan(\theta)$ in the $r = -ct$ relationship. This is exactly what happens in the XTENT MODEL (Renfrew and Level, 1979). The poset changes as we can see that in the example in Fig. C.6 as we reduce c from $c = 1$ the first edge to go will be the (A, C) edge.

We can define the disks formally as follows (T)⁸

$$\Delta(\mathbf{x}, r) = \{\mathbf{y} \in X | d_{\text{disk}}(\mathbf{y}, \mathbf{x}) < r\}. \quad (\text{C.1})$$

$$d_{\text{disk}}(\mathbf{x}, \mathbf{y}) = \max_i \{\mathbf{w}^{(i)} \cdot (\mathbf{x} - \mathbf{y})\}. \quad (\text{C.2})$$

For DISK SPACE see (Suzuki et al., 2019) and references therein. The idea is that we can define “disks” or more generally polyhedra in some d -dimensional metric space X (and more generally in a partial metric space) by specifying the perpendiculars to these boundaries. So working in a d -dimensional space we have a set of finite vectors in this space $\mathcal{W} = \{\mathbf{w}^{(i)}\}$.

⁸(T) IS the following true. The d_{disk} distance is a quasi-metric because the function is not symmetric so $d_W(\mathbf{x}, \mathbf{y}) = d_W(\mathbf{y}, \mathbf{x})$ is not always true.

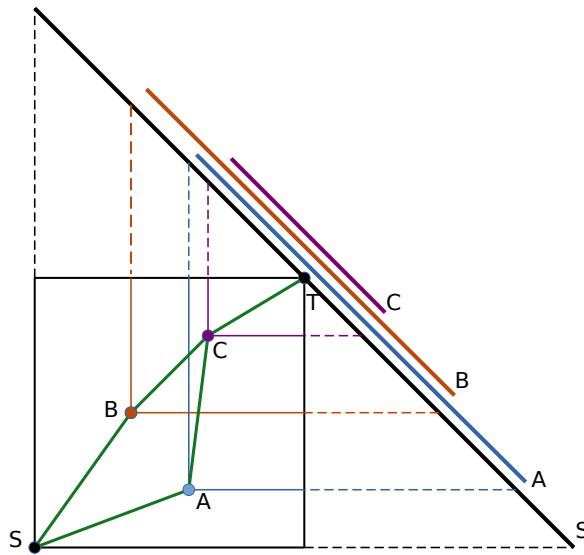


Figure C.6: An illustration of how a cube-space in two dimensions is related to a disk space where disks are in one dimension. The interval runs from the global sink S at $\mathbf{0} = (0, 0)$ to the global sink, the target node T at $\mathbf{1} = (1, 1)$. The interval is the square shown in solid black lines. There are three points inside the interval shown as circles with labels A , B and C . The green line connecting points show the relations after transitive reduction (the Hasse diagram of the poset). The solid lines leaving each point show the boundaries of the forward connection kernel.

The disk space is shown as the projection of the forward connection kernels onto the plane perpendicular to the $(1, 1)$ vector which passes through the target (global sink) node T at $(1, 1)$. The continuation of the boundaries of the forward connection kernels are shown as dashed lines of the same colour as the associated point. Each point corresponds to a disk shown as a one-dimensional line of length equal to its radius. These are the solid lines running diagonally from top left $(0, 2)$ to bottom right $(2, 0)$. The solid black diagonal line shows the disk for the source node. The other disks are displaced slightly in the direction of the $(1, 1)$ vector in order to show them clearly. These are the solid lines with labels and colours matching the corresponding point. The disk for the global sink node T is just a single point and the same circle is used to indicate this.

These are used to form a quasi-metric

$$d_W(\mathbf{x}, \mathbf{y}) = \max_i \{\mathbf{w}^{(i)} \cdot (\mathbf{x} - \mathbf{y})\}. \quad (\text{C.3})$$

The *quasi* refers to the fact that function is not symmetric so $d_W(\mathbf{x}, \mathbf{y}) = d_W(\mathbf{y}, \mathbf{x})$ is not always true. **(T)**Not clear to me why this was defined. This d_W is not used later but perhaps it is the example that enable you to define general disks as polytopes, i.e. closed regions bounded by polygons **(end T)**

C.6 XTent model

This is an interesting example of a pyramid/cone model that has been used in a slightly different way to give a DAG with fewer edges than we would define using the rules we have set out so far. The XTENT MODEL (Renfrew and Level, 1979) was defined to enable archaeologists to get a feeling for how different settlements or regions would be dominated/influenced by settlements in the archaeological record. In the XTent model, every site v has a location and size so site v is represented by the pair $(s_v, \mathbf{v}) \in \mathbb{R}^+ \times \mathbb{R}^2$ where \mathbf{v} is the geographical location of the site and s_v is the size of the site. Assuming simple Euclidean measures of distance, we draw a

This is mapped onto disks $D_i(\theta) = (\mathbf{x}_i, s_i \tan(\theta))$ where usually we have a (quasi)-distance $d(\mathbf{x}, \mathbf{y})$ on the geographical space \mathbb{R}^2 , often chosen to be a Euclidean metric. In our language,

the disks always define a poset and a DAG through the generalised criterion (??). However, the distance provided on the space X obeys the triangle identity, e.g. it is at least a quasi-metric, then we use the metric to define real disks in X and use set inclusion as in the first definition (B.4) but we arrive at the same result. This gives us a set of posets/DAGs $\mathcal{D}(\theta)$ as a function of θ . As $\theta \rightarrow \pi$ we get the sites in a total order based on site size. All the other DAGs are subgraphs of this total order DAG.

C.7 Boxicity

The BOXICITY construction (see TSE `boxicity.tex` notes, perhaps copy in here) reminds me of the interval graphs we encountered when looking at DAGs (directed acyclic graphs) which in turn came out of interest in things like the cube box space.

The following is from Wikipedia on boxicity

In graph theory, boxicity is a graph invariant, introduced by Fred S. Roberts in 1969.

The boxicity of a graph is the minimum dimension in which a given graph can be represented as an intersection graph of axis-parallel boxes. That is, there must exist a one-to-one correspondence between the vertices of the graph and a set of boxes, such that two boxes intersect if and only if there is an edge connecting the corresponding vertices.

C.7.1 Boxicity to DAG

In the boxicity construction, use the centre of the boxes (say) as coordinates of vertices. In each dimension an overlap of two boxes corresponds to one type D links if boxes are D dimensional. We can add a direction for each link based on the coordinate value e.g. from small to large value of a coordinate. This is a “multiplex DAG”, with a different types of edge for each coordinate. Each layer is a DAG though there can be cycles if you move between layers.

The cube box space construction works in the same way but you only have a single type of edge and then ONLY if a vertex has a larger coordinate in every dimension (i.e. all possible edges are present). So here Boxicity to DAG a link is directed created ONLY if every type of directed multiedge is present in the Boxicity construction.

Put another way, the boxicity picture defines an interval graph in each dimension, each layer, each type of edge. These are all very one dimensional graphs (by definition). Not like the clique like picture associated with a perfect community. Rather it implies the underlying structure has an order.

A less restrictive Boxicity to DAG construction/projection is as follows. From this multiplex boxicity to construct a DAG just project a directed edge iff a directed multiplex edge exist in the same direction in all D layers. Alternatively do this projection only requiring all multiedges present to be in the same direction. Perhaps there is no edge in some of the dimensions. These also give a DAG.

Example: Suppose that we have two ranking systems for journals, IF (impact factor) and CA (Combined Altmetric - a social media based score). Suppose for each one we have the ranking for several years. The boxicity diagram is defined by 2 dimensions, IF and CA, with the size of the box related to the max/min values over the period of time. Alternatively, with one year of data, perhaps always connect to the journal with the next higher and the next lower values in each dimension. Interesting our DAGs use a **different** construction where we have the same two axes but the directed edges are only made if one journal was definitely bigger

than a second in **all** dimensions. However we have the formulae and the code to measure the dimension of the space approximately for large numbers of vertices.

Q: Can we reinterpret the boxicity construction to create a DAG and then use James' code to find a dimension?

In trying to write out what I was thinking, I think I came to realise that there are some interesting links and parallels but that boxicity and the sort of cube box space DAG models we have been using are really optimised for different jobs. So the boxes create edges when they overlap. For the box space construction, you only get edges when all coordinates of one vertex are bigger than another point's coordinates. So the cube-space construction comes from the boxicity case if you take the smallest coordinate of each boxes to minus infinity $-\infty$ and add directions. A further difference, with boxicity coordinates you have a natural ordering but in the constructions we have used the direction of the order is important. Here each coordinate can be ordered in 2 different ways so D coordinates gives us 2^D different ways of mapping onto a cube box space. The maps I had in mind were ones which guaranteed no cycles but there are clearly many other less restrictive maps which give a direction to multiedges but contain cycles (though each layer, each dimension is itself a DAG, the cycles are only possible if you change layers/dimensions!).

Is the basic problem that there is no natural directionality in a boxicity diagram? That is we can assign coordinates but there is no natural direction of flow. Its as if we were embedding our boxicity diagrams in a spatial manifold not a space-time one. Again the role of time is crucial. In fact we are using a Euclidean D dimensional space \mathbb{E}^D to define a boxicity diagram so this is not a surprise. Again the role of time is crucial.

We can always add a direction to an undirected network in such a way that we have a DAG. I think we just do a breadth first search from any set of vertices and the time taken to reach a vertex sets its time coordinate (its height in the DAG). If edges connect vertices with the same time, I think we can perturb the times in a consistent manner to get a DAG (but not sure about that, might have to delete such edges). However this is not a natural time coordinate, we could do this from any point in any space with a Euclidean metric.

So surely people have come up with estimates of spatial dimensions for networks before, for instance Song et al. (2007), Gallos et al. (2007)? Yes and some of the methods use a breadth first (burning) algorithm. That is if my network is embedded in a space with a Euclidean metric (if not a full Euclidean space) can I not do some simple box counting? So look at how the number of nodes ℓ steps away grows.

C.7.2 Embedding a graph in space

How do we give the vertices of a graph spatial coordinates, be they box coordinates in boxicity or a single point in the Serrano et al. (2008), Boguñá et al. (2010)? I am still intrigued by the link between the boxicity models and the \mathbb{H}^2 hidden variable models of Boguñá and collaborators Serrano et al. (2008), Boguñá et al. (2010). Essentially the hidden variable in the Boguñá et al. model would translate to the dimension of the box for each point. Boxicity though uses Euclidean \mathbb{E}^D while Serrano et al. (2008), Boguñá et al. (2010) use the two-dimensional hyperbolic space \mathbb{H}^2 .

To bring boxicity closer to the models of Serrano et al. (2008), Boguñá et al. (2010) we need spheres not boxes, a sphericity model(?). So you connect points $i, j = 1, 2, \dots, N$ with hidden variables κ_i and κ_j so that the adjacency matrix is

$$A_{ij} = \theta(\kappa_i - r_{ij})\theta(\kappa_j - r_{ij}) \quad (\text{C.1})$$

where r_{ij} is the distance between the vertices/points $i, j = 1, 2, \dots, N$, e.g. distance measured as in \mathbb{H}^2 for Serrano et al. (2008), Boguñá et al. (2010). Here the $\theta(x)$ can be seen as a limiting

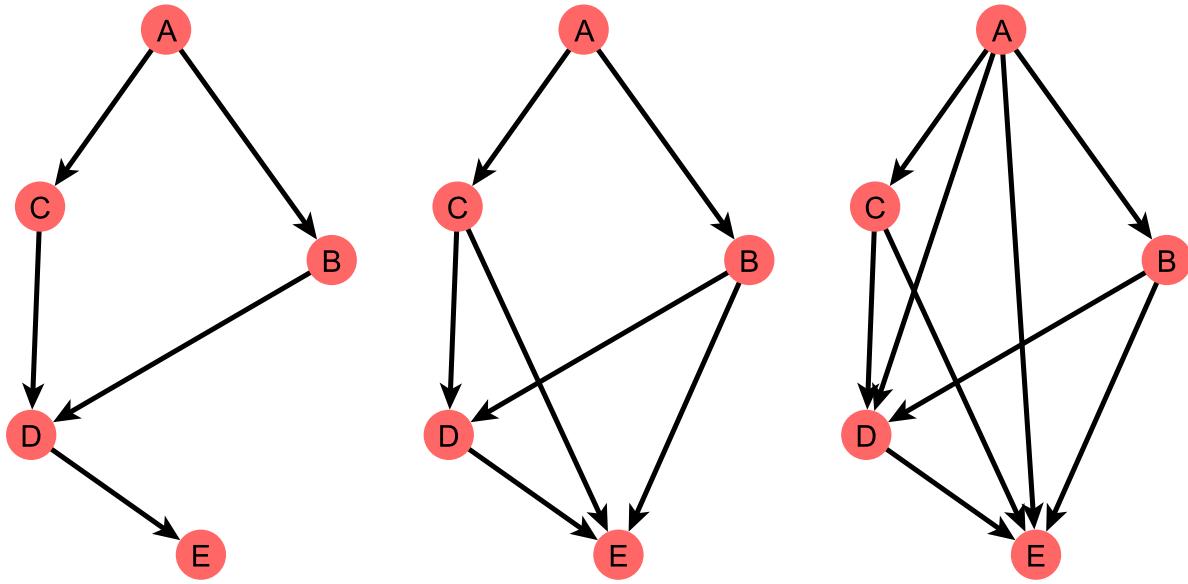


Figure C.7: Example of transitive reduction (left) and transitive completion (right) of the Directed acyclic network in the centre.

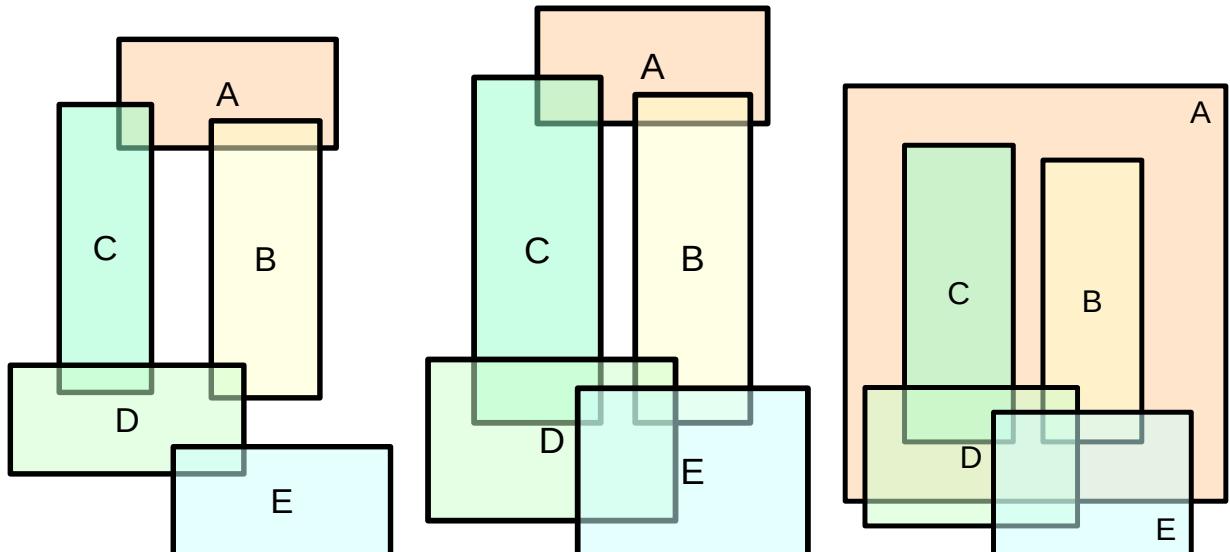


Figure C.8: Example of how toxicity constructions are mapped onto DAGs. These are the toxicity diagrams for the DAGs shown in (C.7).

form of the monotonically decreasing function used in the papers though they use powerlaw forms and I'm not sure there is a simple link between the two forms. Serrano et al. (2008), Boguñá et al. (2010) use something like

$$p_{ij} = \frac{\kappa_i \kappa_j}{f}(r_{ij}) \quad (\text{C.2})$$

as the probability of a link in some ensemble of graphs where f is monotonically decreasing. When f is a power law its seems to allow them to link κ to the expected degree of a node.

Boxicity in this language assigns a vector of hidden variables κ_a^i where $a = 1, 2, \dots, D$ (so we have a D dimensional space to each point i (the size of the box in each dimension) and working in \mathbb{E}^D Euclidean metric space it assigns an edge between two points i and j so that the adjacency matrix is

$$A_{ij} = \prod_a \theta(\kappa_a^i - x_a^i) \theta(\kappa_a^j - x_a^j) \quad (\text{C.3})$$

where x_a^i is the a -th coordinate of vertex i .

One thing to note is that Serrano et al. (2008), Boguñá et al. (2010) have a programme which assigns these hidden coordinates because they link their hidden variables κ_a to the degree of a node. To do this we need the probability of assigning p_{ij} a link between i and j to look like

$$p_{ij} \propto \prod_a \frac{1}{1 + \exp(\beta(\kappa_a^i - x_a^i))} \frac{1}{1 + \exp(\beta(\kappa_a^j - x_a^j))} \quad (\text{C.4})$$

We would then vary the coordinates x_a^i to optimise the log likelihood of a given real network defined by some data set coming from this statistical model. For Boximity, $\beta = \infty$ gives the theta function form but this β allows for imperfect matchings. Still leaves with trying to estimate the dimension. See Krioukov et al. (2009).

C.7.3 Further ideas

The eigenvectors (of adjacency, Laplacian or random walker transfer matrix) all give every vertex a coordinate, one for every eigenvector. Suppose we use the eigenvectors of the largest m -eigenvalues to assign coordinate in an m -dimensional space. We can now construct a boximity style multigraph or a cube box space DAG?

BTW if you are interested about the connection of boximity and topology/genus of a graph: Esperet and Joret (2012). Boximity of graphs on surfaces. Graphs and Combinatorics,

C.7.4 Comments

TSE Comment: Put another way, the boximity picture defines an interval graph in each dimension, each type of edge. These are all very one dimensional graphs (by definition). Not like the clique like picture associated with a perfect community. Rather it implies the underlying structure has an order. CL (Chuan Wen, 2015) Answer: A clique/complete graph has boximity = 1, i.e. it is an interval graph (1 dimensional). However the box representation of a graph is not unique as the intervals of a clique can be identical (hence trivially overlapping) OR there is some ordering/structure with the condition that all the intervals overlap. Thus the box representation of a graph often is not meaningful, but its boximity determines the minimum number of metrics to "measure" the graph.

TSE Comment: Tell me how the boximity construction is related to Minkowski space. CL Answer: Page 21 in my thesis (Chuan Wen, 2015). The complete proof is in "L.C. Freeman. Spheres, cubes and boxes: Graph dimensionality and network structure" (Freeman, 1983).

Freeman not only proved the connection between boxicity and Minkowski space, he had also studied the connection when the vertices are spheres instead of boxes.

TSE Comment: Can we reinterpret the boxicity construction to create a DAG and then use James' code to find a dimension? CL Answer: So what you are proposing is to assign directions to an undirected graph (in a meaningful way) such that the number of cycles are minimized?

C.8 Application of Box Spaces

The cube-space rules have been used to ordered items with a given rating or ranking (e.g. Mathematicians in Chen et al. (2017), artists from Kitromilidis and Evans (2017) but not in paper, Greek gods of Fig A2 in Vasiliauskaite and Evans (2020)).

The cube-space rules have been used for image categorisations where there is a DAG built up of concepts/terms from captions/labels linked to the images (Vendrov et al., 2016).

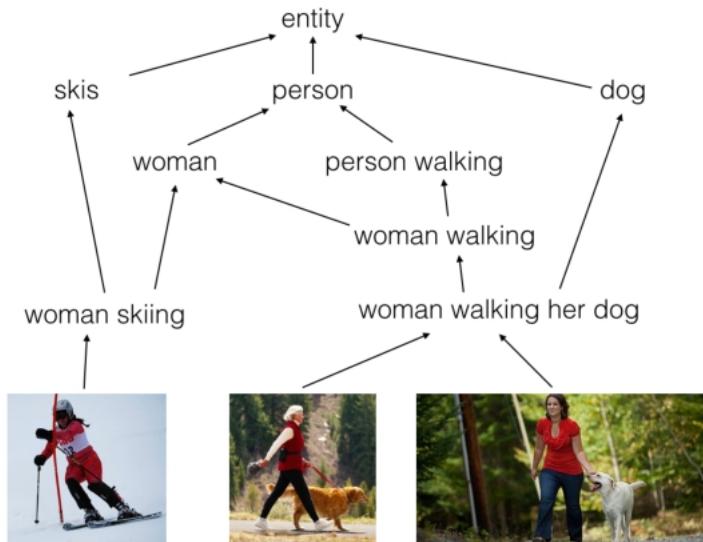


Figure 1: A slice of the visual-semantic hierarchy

Figure C.9: Figure 1 from Vendrov et al. (2016) showing how images are ordered as a DAG (here shown transitively reduced as a Hasse diagram) using terms from captions or labels or general concepts such as “entity”.

Index

- (...), 207
 L_p metric, 216
 \cap , 206
 \cup , 206
 $d^{(lp)}$, 41
 $d^{(sp)}$, 20
 ℓ , 17, *see* length
 \in , 206
 ω , 19
 \prec , 35, 39, 56
 \preceq , 35, 39, 56
 \setminus , 206
 σ -algebra, 217
 \subset , 206
 \subseteq , 206
 $\{\dots\}$, 206
 b , *see* betweenness
 c , *see* closeness
 $d^{(DAG)}$, 123
 $d^{(lp)}(v, u)$, 108
80–20 rule, *see* Pareto principle

academic papers, 144
acausal path, 18, 160
acyclic, 18
adjacency list, 169
adjacency matrix, 14, 53
numerical, 168
agony, 181, 185
algorithm
 topological sort, 44, 45, 171
 total order, 44
 total sort, 171
ancestor, 37
antichain, 61, 62, 63
array, 168
 C++, 207
 dynamic, 168
attachment list, 177
authority value, 98
average neighbour value, 90, 90

average shortest path length, 20
bag, *see* multiset
ball
 generalised, 209
Barabási-Albert model, 130
Bayesian belief network, 157
Bayesian network, 159
belief network, *see* Bayesian network
Beta function, 138, 205
betweenness, 22, 75
binomial distribution, 29, 208
Boost graph library, 167
box space, 217, 219
 metric, 221
boxicity, 226
broadcast, 24
broadcast path count, 81, 82
broadcast process, 24

C++
 containers, 207
 map, 168
 standard template library, 168
 vector, 168
causal inference, 158
causal path, 17, 160
causal set, 212
causal set theory, 123
central limit theorem, 133
centrality, 15, 21, 21, 23, 72, 77, 152, 153, 180
 average neighbour value, 90
 betweenness, 22
 broadcast path count, 81, 82
 closeness, 22, 77
 degree, 22
 diffusion path count, 84–86
 eigenvalue, 81
 in-degree, 73
 out-degree, 73
chain, 59, 59

- maximal, 61
- minimal element, 60
- chain cover, 61, 61
- child, 35
- Chung-Lu random graph, 133
- citation network, 129, **142**, **144**, 144
 - acausal edge, 181
- citations, 144
- clique, 211, 212
 - cover, 212
- clique cover, 212
- closeness, 22, 77
- co-authorship network, 28
- collider, 160
- colour power set, 58
- communities, 212
- comparable, 56
- complex network, 27
- component, 11, 21
- computer datasets, **146**
- computer languages, 207
- computer package dependency, **148**
- cone space, 122, 128
- cone-space DAG, 212
- cone-space model, 122
- configuration model, 30, 173
- confounder, 160
- connected
 - strongly, 21
 - weakly, 21
- connection kernel, 116
- contrastatus, 77, 87, 153
- core, 212
- cover, 207
- criticality, 43, 100, 101, 104
- cube space
 - order embedding, 129
- cube-space DAG, 116, **116**, **157**, 214
- cube-space model, **116**, 213
- cube-space rule, 116, 213
- cumulative advantage, 129
- cycle, **16**, 17, 18
- DAG, *see* directed acyclic graph
 - Chung-Lu random graph, 133
 - random circuit model, 133
- DAG metric, 108, 123, 215
 - symmetric, 108
- DAGlet, **63**
- de Solla Price, *see* Price
- defective matrix, 202
- degree, 15
 - distribution, 15
 - enhanced, 73
 - in-, 15, 73
 - out-, 15, 73
 - reduced, 73
- degree distribution, 15
- depth, **41**, 41, 42, 43, 101, 179, 179
- descendant, 37
- descendent, 35
- DFS, *see* depth first search
- diameter, 107
- dictionary
 - Python, 168
- difference equation, 134
- difference equations, 204
- diffusion path count, 84–86
- diffusion process, 25
- digraph, *see* directed network, **14**
- Dilworth’s theorem, 62
- dimension, 219
- directed acyclic graph, **33**, 33
 - Bayesian network, **157**
 - causal inference, **157**
 - computer datasets, **146**
 - computer package dependency, **148**
 - cone-space DAG, 212
 - cube space model, **116**
 - cube-space DAG, 213
 - DAG of DAGs, 48
 - disk space, 124
 - edge swap, 174
 - Erdős-Réyni random DAG, 111
 - height, 43, 61, 101
 - influence, 126
 - metric, 108, 123, 215
 - Minkowski space-time DAG, 120
 - programming language, **147**
 - pyramid DAG, 124
 - random order DAG, 111
 - rating, **156**
 - reverse, 87
 - set inclusion, 47
 - set inclusion DAG, 112
 - weighted, 94
- directed acylic graph
 - disk space DAG, 114
- directed network, 13, **14**
- directed path, 18
- directed random geometric graph, 121

- directed tree, 63
 directed trees, 67
 disk, 113, 209
 disk space, 113, 124, 209, 213, 224
 disk space DAG, 114
 distance, 19, 19, 20, 105, 118
 - Euclidean, 107
 - height, 107
 - Minkowski, 216
 - pseudo, 41
 dynamic array, 168
- edge, 11, 12
 - acausal, 181
 - directed, 14, 207
 - essential, 48, 109
 - head node, 13, 14
 - hyperedge, 12
 - inessential, 48, 109
 - network length, 108
 - reciprocated, 14
 - self, 53
 - self loop, 13
 - source node, 13, 14
 - stub, 15
 - tail node, 13, 14
 - target node, 13, 14
 - unreciprocated, 14
 - weight, 14, 14, 108
 edge list, 170
 edge weight, 14, 95, 102
 eigenvalue centrality, 81, 83, 94
 electrical circuit, 92
 element, 206
 enhanced degree, 73
 ensemble average, 133
 Erdős, Paul, 27
 Erdős-Réyni random graph, 111
 Erdős, Paul, 28
 essential edge, 48, 109
 Euclidean distance, 107
 Euclidean metric, 118
 factorial moments, 204
 farness, 78
 fat-tail, 16
 food web, 151
 formal ball, 210
 future, 37
 future closeness, 77
 future neighbourhood, 35, 36
- Gamma function, 136, 204
 generalised ball, 209
 generating function, 203
 geodesic, 106–108
 geometric set inclusion DAG, 113
 GIS
 - geographical information software, 127
 graph, *see* network
 graph Laplacian, 26
 graph theory, 12
 graphical causal model, 34
 graphlet, 211, 212
 Gromov product, 222
 ground set, 56
 Guare, John, 20
- Hasse diagram, 49, 50, 57, 59, 113, 116, 157, 178
 head, 13, 14
 height, 41, 41, 43, 46, 101, 179, 179
 - DAG, 61
 - distance, 107
 height distance, 107
 height of the layering, 109, 110
 hierarchy, 47, 77, 87, 153, 185
 higher order structure, 12
 HITS centrality, 97, 98
 homogeneous function, *see* scale invariant function
 homogeneous partially ordered measure space, 219
 HPO-space, 219
 hubs value, 98
 hyperedge, 12
 hypergraph, 12
 Hyperlink-Induced Topic Search, *see* HITS
 identity of indiscernibles, 106, 108
 in-degree, 15, 73
 inclusion, 58
 - set, 206
 incomparable, 56
 indiscernibles, 106
 induced subgraph, 211, 212
 inessential edge, 48, 109
 inessential edges, 48
 influence DAG, 126
 intersection, 206
 interval, 37, 38
 interval DAG, 80
 Java, 207

- collections, 207
- list, 168
- map, 168
- set, 208
- Kahn's algorithm, 170
- Katz centrality, 84, 94
- keystone index, 71, 72, 87, 153
- Kronecker delta, 23, 27
- labelled graph, 64
- Laplacian, **26**, 26, 92, 93, 182
- layer, 109
 - width, 110
- layering, 109
- length, 17
 - average shortest path, 20
 - network, 17
 - path, 17
 - path weight, 19
- Leontief Input-Output model, 94
- line graph, 68, *69*
 - directed, *69*
- linear algebra, 201
- linear order, *see* total order
- linear process, 201
- link, 11, 12
- list
 - Java, 168
 - Python, 168, 207
- longest path, 39, *40*
 - length, 41
- longest-path metric, 108
- lower triangular matrix, 53
- lowest common ancestor, 105
- L_p metric, 216
- Möbius inversion, 54, *55*
- main path, 102
- main path analysis, 101, *104*
- map
 - C++, 168
 - Java, 168
- Markov process, 134
- master equation, 133, 134
- matrix
 - adjacency, 14
 - lower triangular, 53
 - nilpotent, 53
 - stochastic, 26
 - strictly triangular, 53
- transfer, 84
- triangular, 52
- Matthew effect, 129
- maximal antichain, 61, *62*
- maximal chain, 60
- maximal element, 56, 60
- maximum chain, 61
- mean field approximation, 133
- measure, 217, 218
- measure space, 115
- member, 206
- metadata, 180
- metric, 106, 118
 - L_p , 216
 - box space, 221
 - DAG, 108, 123
 - longest-path, 108
 - Lorentzian, 123
 - symmetric DAG, 108
- Milgram, Stanley, 20
- minimal element, 56, 60
- Minkowski distance, 216
- Minkowski space-time, 212
- Minkowski space-time DAG, **120**, 120
- Mirsky's theorem, 63
- mixed trophic impact, 71, 72, 94, 154
- mixed trophic index, 96
- mixed tropic index, 96
- model
 - Chung-Lu random graph, 133
 - configuration model, 30
 - cube-space DAG, 116, 213
 - Erdős-Réyni random graph, 111
 - geometric set inclusion DAG, 113
 - Minkowski space-time DAG, 120
 - null, 27
 - Price model, 129
 - random circuit model, 133
 - random order DAG, **111**, 111
 - set inclusion DAG, 112
- moments, 204
- Moore-Penrose inverse, 183
- moral graphs, 70
- motif, 66, 211, 212
- MPA, 101
- multiedge, 132, 134
- multiple path centrality, 75, 81
- multiset, 207
 - C++, 208
- neighbourhood, *36*

- nested hierarchy, 47
net status, 78
network, 11, 11
 Bayesian, 157
 citation, 129, 142, 144
 computer datasets, 146
 computer package dependency, 148
 directed, 13, 14
 directed acyclic graph, 33
 hypergraph, 12
 karate club, 180
 labelled, 64
 programming language DAG, 147
 random, 30
 regular, 15, 29
 signed, 86
 simple, 13, 14
 temporal, 68
 tree, 18
 unlabelled, 64
 weakly connected, 14
 weighted, 14, 14
network Laplacian, 92
network length, 17, 19, 51, 108, 109
NetworkX, 167
nilpotent, 53
node, 11, 12
 ancestor, 37
 centrality, 15, 21, 23, 77, 87, 152, 153, 180
 child, 35
 criticality, 43, 100, 101
 degree, 15
 depth, 43, 101
 descendant, 37
 descendent, 35
 head, 13, 14
 height, 43, 46, 101
 parent, 35
 reachable, 17
 sink, 36
 source, 13, 14, 36
 status, 77
 strength, 95
 tail, 13, 14
 target, 13, 14
nodes
 see node, 11
non-negativity, 105, 108, 124, 215
null model, 27
OEIS, 64
open interval, 38
optimal path centrality, 75, 80
order
 linear, 58
 partial, 55
 topological, 43
 total, 42, 58
order embedding, *see* cube space
ordered random geometric graph, 118, 120
ORGG, *see* ordered random geometric graph
oriented tree, 63
out-degree, 15, 73
PageRank, 84, 85, 202
parent, 35
Pareto principle, 129
partial order, 55, 56, 59
partial subgraph, 211, 212
partially ordered measure space, 219
partially ordered set, *see* poset
partition, 21, 58, 207
partition function, 203
past, 37
past closeness, 77
past neighbourhood, 35, 36
patents, 144
path, 16, 17, 18, 39, 39, 207
 acausal, 18
 causal, 17, 160
 causal inference, 160
 cycle, 18
 distance, 20
 geodesic, 107
 length, 17
 longest, 40, 75
 network length, 17
 shortest, 20, 75
 trivial, 17, 19
 weight, 19
path entropy, 102
Perron-Frobenius, 99
Perron-Frobenius theorem, 99
Poisson distribution, 29, 208
Poisson point process, 107, 114, 115, 214
polytree, 63
poset, 55, 56, 57, 219
 Möbius inversion, 54
power set, 48, 58, 217
 colours, 58
power-law distribution, 129

- PPP, *see* Poisson point process
 predecessor, 35, 36, 116
 preferential attachment, 130
 Price model, 129
 degree evolution, 139
 longest path, 140
 master equation, 134
 numerical algorithm, 177
 Price, Derek de Solla, 129
 prior art, 144
 programming language DAG, 147
 proper subset, 206
 proper time, 121
 pseudo-distance, 41
 pseudo-inverse, 183
 pyramid DAGs, 124
 Python
 dictionary, 168
 list, 168, 207
 set, 208
- random
 uniformly at, 15
 random circuit model, 133
 random geometric graph, 118, 119
 ordered, 118
 random geometric network, 107
 random network, 30
 random order DAG, 111, 111, 112
 rating, 182
 rating DAG, 156
 reachable, 17, 35, 39
 reciprocated, 14
 reduced degree, 73
 reflexivity, 105, 108, 124, 215
 regular network, 15, 29
 Relation to antichains?, 109
 relative complement, 206
 reverse DAG, 87
 reverse metric, 221
 box space, 221
 reverse triangle identity, 108, 124, 215, 216
 RGG, *see* random geometric graph
- s:betweenness, 75
 scale invariant function, 138
 scale-free behaviour, 138
 self edge, 53
 self loop, 13
 semiwalks, 160
 sequence, 207
- set, 206
 C++ STL, 208
 cover, 207
 element, 206
 inclusion, 58, 206
 intersection, 206
 Java, 208
 partition, 58, 207
 power, 58
 Python, 208
 subtraction, 206
 set inclusion DAG, 47, 112, 112, 113, 114
 shortest path, 20, 75
 signed network, 86
 Simon's model, 130
 simple network, 13
 sink node, 36, 36
 six-degrees of separation, 20
 SNA, *see* social network analysis
 social network analysis, 12
 sociogram, 12
 source node, 13, 14, 36, 36
 spaghetti monsters, 178
 SpringRank, 91
 springrank, 182
 standard template library, 168
 multiset, 208
 set, 208
 status, 77, 87, 153
 STL, *see* standard template Library
 stochastic matrix, 26, 95, 154
 strength, 95
 strictly lower triangular matrix, 53
 strictly triangular matrix, 53
 strongly connected, 21
 stub, 12, 15
 subgraph, 13
 induced, 211, 212
 partial, 211, 212
 subset, 206
 proper, 206
 successor, 35, 36
 symmetric DAG metric, 108
 symmetry, 105
- tail, 13, 14
 target node, 13, 14
 temporal network, 68, 68
 topological order, *see* total order
 topological sort, 43, 45, 170, 170

total order, **42**, 42, 45, 58, 60, 68, 110, 170, 174

trail, 18

transfer matrix, 25, 84

transitive completion, **46**, 46, 73

transitive reduction, **46**, 47, 73

transitivity, 56

tree, 18, 18, 47, **63**, 185

triangle identity, 106, 216

reverse, 108

triangle inequality, 106, 118

triangular matrix, **52**, 52, 112

trivial path, 19

trivial walk, 17

trophic cascade, 152

trophic network, *see* food web

uniformly at random, 15

union, 206

unlabelled graph, 64

upper triangular matrix, 53

vector

C++, 168

visualisation, *43*, 46, 101, **178**

walk, **16**, 16, 17, 33, 39

edge, 17

length, 17

network length, 17

number of, 19

semiwalk, 160

weight of, 19

weakly connected, 14, 21

weight, 14, 14

weight of a walk, 19

weighted DAG, 94

weighted network, 14

width, 62, 62

XTent model, **126**, 126, 129, 224, 225

Yule-Simon distribution, 130

Zachary karate club network, 180

Zipf plot, *131*, 145