Making Communities Show Respect for Order

V. Vasiliauskaite[1], T.S. Evans

Centre for Complexity Science, and Theoretical Physics Group,
Imperial College London, SW7 2AZ, U.K.
30th August 2019

Abstract

In this work we give a community detection algorithm in which the communities both respects the intrinsic order of a directed acyclic graph and also finds similar nodes. We take inspiration from classic similarity measures of bibliometrics, used to assess how similar two publications are, based on their relative citation patterns. We study the algorithm's performance and antichain properties in artificial models and in real networks, such as citation graphs and food webs. We show how well this partitioning algorithm distinguishes and groups together nodes of the same origin (in a citation network, the origin is a topic or a research field). We make the comparison between our partitioning algorithm and standard hierarchical layering tools as well as community detection methods. We show that our algorithm produces different communities from standard layering algorithms.

1 Introduction

Nodes in networks have many natural orders. Every centrality measure allows us to say if one node has a higher centrality value than another. However, some systems can have important constraint leading to a characteristic order in a network; examples include the publication dates of papers in a citation network, dependency of packages in computer software, and predator-prey relationships in a food web. If edges respect this order, they exist only if they link a high value node to a lower value node, from an earlier paper to a later paper, then edges are directed and there can be no cycles — a Directed Acyclic Graph (DAG).

A link between two nodes in a DAG must always encode the *order* of the pair. It is the converse which is important here: certain links can not exist because of the inherent order. This means that two nodes can be very similar but they can not be connected while in standard network analysis such an edge would be assumed to exist most of the time in a DAG. For example, two papers can be produced independently at the same time with similar new results yet by definition they can not cite each other. The development of the relativistic model for the Higgs mechanism is a good illustration of this as it is attributed to three independent groups: Brout and Englert (August 1964), Higgs (October 1964), and Guralnik, Hagen and Kibble (November 1964). Only last of these three papers cites the two earlier publications but there is no citation between the first two.

So in order to understand DAGs, we need to adapt our standard network analysis tools to make them respect the order implicit in these common network topologies. In this paper we will focus on the important topic of community detection in networks, clustering in the language of data science, in which the aim is to find sets of similar nodes [1, 2]. Since most network methods assume that a link between nodes indicates similarity, the communities found are subgraphs where there are more links within the community than there are to the rest of the network [1]. This is justified when links indicate node similarity that is not necessarily true

---

[1]Corresponding Author.

for DAGs so we need to find new ways to define clusters otherwise the order in a DAG may obscure some of the natural clustering.

To see how we will find communities in a DAG, we start by noting that one expression of the order in a DAG is that there is a natural hierarchy in the system. Two computer packages which fill a very similar role will not depend on each other but they will draw on similar "lower level" packages reflecting a hierarchy and an order in packages. For instance, the python network package `networkx` is a prerequisite for two python community detection algorithms, `python-louvain` and `demon`. In the Florida-bay foodweb we study in Section 4.3, pinfish, parrotfish and manatee are grouped together because they feed on similar species, such as detritivorous polychaetes, sponges, bivalves. So in a DAG a natural property of any nodes at the same level in the hierarchy is that they are *not* connected, directly by an edge or even indirectly via a longer path. So, in the context of a network with an order, with a hierarchy, it is very natural to cluster nodes which are not connected by any path, and these are called ANTICHAINS, see Fig. 1. So in our approach, in order to respect the order inherent in a DAG we will create communities which are antichains.

The next problem is that there are many possible antichains and each node can be in many different antichains. Also nodes in useful communities will need to be similar in some sense. In a set of computer packages, we don't want to cluster a package on games with one on networks, rather we want to collect all the different network packages together in one community. Likewise, we don't want to cluster species at the same level in food webs if they are from completely different environments. So we will aim to find the antichains which contain nodes which are similar by some appropriate measure. We take our inspiration from classic measures used to assess the similarity of two documents from their citation network alone. In *bibliographic coupling* the similarity of two documents is measured using the overlap of their bibliographies [3]. The *co-citation* similarity of two documents [4] uses the overlap in the citing documents. So by looking into similarities of neighbourhoods of two nodes, we can say something about how similar they are themselves.

Our aim in this paper is to produce a method to find communities in a DAG which take account of the sense of order, the hierarchy, implicit in a DAG. We partition the nodes of a DAG into antichains which have large neighbourhood overlaps. We will also look at the properties of such DAG clustering and how it relates to other types of DAG layering and clustering. Lastly we investigate the potential uses of this approach to study citation networks and food webs.

Our method is related to two different types of network algorithms: community detection and DAG layering. Community detection does not include a notion of "layers". As a consequence, a conventional community can be composed of nodes from many different layers. DAG layering, on the other hand, does not have a notion of similarity, so a layer is composed of hierarchically equivalent nodes but a layer does not represent a community. Our algorithm bridges this gap and finds layer decomposition of a DAG with a notion of similarity.

Community detection is an active research area in network science. This field is wide and many different approaches to finding groups of similar nodes were proposed. An extensive discussion of the main approaches to community detection in networks is given in [1]. Some works have studied the stability and performance of conventional community detection algorithms in DAGs such as [5, 6]. There is also some work on specialised approaches to community detection in directed acyclic graphs, for instance [7], in addition to clustering algorithms developed for analysis of task scheduling graphs (which are also DAGs), for example see [8] and references therein.

Graph layering is another topic related to our work. Graph layering is used for drawing
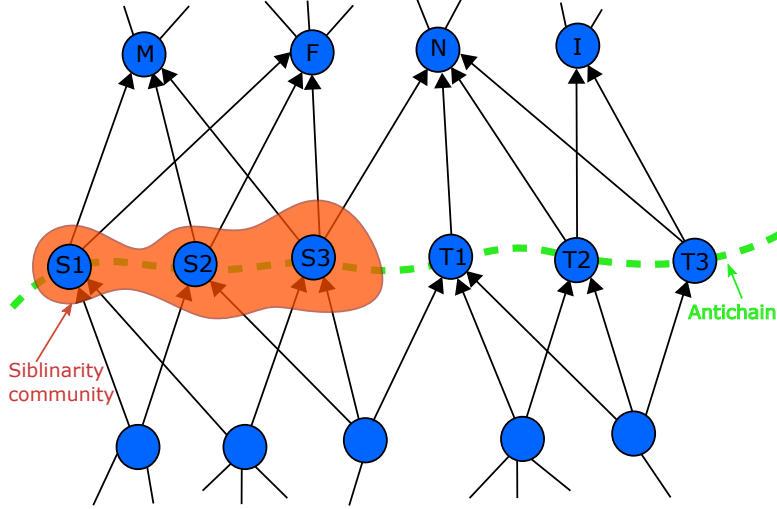
Figure 1: An antichain is a subset of nodes in the graph, such that none of the nodes are pairwise connected with edges or paths. In this illustration, an antichain is composed of nodes $\{S_1, S_2, S_3, T_1, T_2, T_3\}$ (the green dotted line is drawn as a guide of eye). Although in the same antichain, not all nodes share many neighbours. Nodes $\{S_1, S_2, S_3\}$ share many successors (they are all connected to nodes M, F), so we join them in a Siblinarity community.

hierarchical graphs, such as trees and DAGs. Sugyiama graph drawing algorithms use various layering techniques to remove edge overlap as much as possible[9]. The longest-path algorithm is used to find a layering with the smallest number of layers[10]. More sophisticated methods aim to limit the number of layers in the graph, variation in the size of each layer, number of layers with crossing edges or maximum width of any of the layers [11, 12, 13, 14, 15, 16].

Our aim here is find a clustering algorithm which both respects the topology and ordering a DAG while ensuring the clusters only contain similar nodes. We start by by discussing the methods we use in section 2. We look at the relevant properties of DAGs in section 2.1. We review a standard way to partition a DAG by considering the heights and depths of nodes in section 2.2. We then describe the algorithm we used to find our siblinarity communities in section 2.3 and section 2.4. In section 3 we describe the models and data used to test our algorithm. The results of our analysis are given in section 4 wehere we study the properties of siblinarity-based communities, comparing them to alternative methods and exploring variations of our algorithm. We conclude with a brief overview of our work in section 5. Further details and additional examples are contained in an appendix. We provide data for many of our examples online [17] while the source of data for any remaining examples is given in our bibliography.

2 Methods

2.1 Basic DAG properties

Suppose we have a DAG (directed acyclic graph) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the set of nodes and edges in the graph are $\mathcal{V}$ and $\mathcal{E}$ respectively. We will use $N = |\mathcal{V}|$ for the number of nodes in the DAG. We will denote an edge from node n to node m as (n, m).

A PATH in any directed graph, including a DAG, is a sequence of nodes in which consecutive nodes are linked by an edge in the correct direction [18]. The length of the path is the number

of edges in that path, one less than the number of nodes in the path. This allows us to define an ANTICHAIN $\mathcal{A}$ as a set of nodes in which there is no path between any of the nodes in the antichain. We will later use the antichains in both the original DAG and in directed graphs we shall derive.

The lack of cycles in a DAG leads to a special property, namely that every DAG is linked to a unique PARTIAL ORDER on the set of nodes. That is we can say $n \prec m$ if there is a path *from n to m*. Note there need not be any sense of order between two nodes, i.e. when there is no path between the nodes, it is this partial order which we aim to respect in our community detection. This mathematical property of the DAG's partial order is often seen in key properties of a real data set. For instance, in a perfect citation network, the publication dates of documents express the partial order. That is if $n \prec m$ for two documents n and m in the citation network then we know that document n must be published before document m where we take the edges in the DAG to run *from* the older document being cited *to* the bibliographies of newer documents. The partial order in this case reflects the "arrow of time" inherent in the publication of documents. In computer libraries, each package tends to require less specialised packages. For a food web, the mathematical partial order may reflect the tendency of larger animals to eat only smaller ones.

In general, this sense of order in a DAG is often seen as a natural hierarchy. For a library of computer packages, we might talk about a low-level package. In a food web, we have "top-level predators". Typically, elements at the same level in the hierarchy are going to have no connections between them since they do the same job at this level, they are alternatives to each other. That is there is a good chance that such elements with no connections are similar and it is the order in the DAG, the level of the hierarchy is telling us about this similarity. In terms of the formal properties of the DAG, these elements form an antichain.

For example, predators of a similar size will not generally eat each other and so are often found at the same level in a food web. Yet such animals could be similar in the sense they compete for similar (smaller) prey. In the food web example we will consider in 4.3, sharks, tarpon and grouper will be found in the same siblinarity community, because they share many common prey, such as killifish and crabs.

2.2   Height and Depth Antichain Partitions

So the approach we take is to look for communities of similar nodes in a DAG which are antichains, denoted as $\mathcal{A}$. In that way, none of the nodes in one of our antichain communities are ordered before or after any of the others in the same community. There is no direct connection between nodes in our antichain communities by definition so these are very different communities from those produced in general networks such as discussed in [1].

For simplicity, we will also restrict ourselves to the case where the each community is an antichain, and the set of all communities is a partition $\mathfrak{A}$, which we call an ANTICHAIN PARTITION. That is every node is in exactly one element (one community) of our antichain partition $\mathfrak{A}$. This is hard or non-fuzzy clustering in the language of data science.

Our first two examples of antichain partitions are well known features of any DAG. We start by noting that another property of any DAG is that we can always assign a HEIGHT to every node. The height $h(n)$ of node n is the length of the longest path to n from any node with zero in-degree. It is straightforward to see that each node on a path must have different heights with the height increasing as you move along the path. Conversely, nodes of the same height cannot have any path between them so that nodes of the same height form an antichain. Thus

we can define the HEIGHT PARTITION $\mathfrak{A}^{(\mathrm{h})}$ to be the set of antichains $\{\mathcal{A}_a^{(\mathrm{h})}\}$, each of which contains all the nodes of a give height

$$\mathfrak{A}^{(\mathrm{h})} = \{\mathcal{A}_a^{(\mathrm{h})}\}, \quad \mathcal{A}_a^{(\mathrm{h})} = \{n|n \in \mathcal{V}, h(n) = a\}. \tag{1}$$

Similarly, we can define the DEPTH $d(n)$ of a node n to be the length of the longest path from n to a node with zero out-degree. Nodes with the same depth are guaranteed to form an antichain so we can define the DEPTH PARTITION $\mathfrak{A}^{(\mathrm{d})}$ to be partition of the set of nodes by their depth,

$$\mathfrak{A}^{(\mathrm{d})} = \{\mathcal{A}_a^{(\mathrm{d})}\}, \quad \mathcal{A}_a^{(\mathrm{d})} = \{n|n \in \mathcal{V}, d(n) = a\}. \tag{2}$$

2.3 Siblinarity Antichain Partitions

Often the *elements* of either the height or depth antichain partitions (an element here is one community, one antichain in a partition) provide one definition of a level in the hierarchy. However, while these antichains respect the order of the DAG, we want to highlight much smaller groups which contain nodes which are much more similar than just the similarity imposed by the order as encoded by an antichain. All the nodes at one height, or those at one depth need not be very similar in general.

To add similarity to the hierarchy constraint encoded through our restriction to antichains, we can take inspiration from classic similarity measures used in bibliometrics. In that context, one way to assess the similarity of two publications is to look at overlap of their neighbours in the citation network. The more two publications share the same neighbours, the more similar they are said to be. The size of an intersection between two paper's bibliographies is called BIBLIOGRAPHIC COUPLING [3], whereas the size of an overlap between articles that they were referenced by is called CO-CITATIONS [4]. So by looking into similarities of neighbourhoods of two nodes, we can say something about how similar they are themselves.

A family tree provides a good example where people are the nodes and edges are from a parent to a child and so point forward in time in terms of birth date. There will be many people in a single generation but, by definition, none will be a parent or a child of any other person in the same generation so each generation forms an antichain. Generations are layers in a natural hierarchy for this DAG. However, almost all people in one generation will have little genetic biological relationship to each other so this large antichain, a single generation, may not be very interesting in many problems. However, if we also look for clusters of people within this generation, people who have common predecessors and so share one or two parents, then these smaller communities may be of more interest. Using such a predecessor similarity measure on top of an antichain constraint would mean a community detection method would be producing communities of siblings.

These ideas of antichain and neighbour similarity are encoded in a function which measures the quality of a given partition $\mathfrak{A}$ of our DAG into antichains, $\mathcal{A}$. Motivated by the family tree example, we call our function SIBLINARITY $S(\mathfrak{A})$ and we define it to be

$$S(\mathfrak{A}) = \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n \in \mathcal{A}} \sum_{m \in \mathcal{A} \backslash n} (\mathrm{sim}(n, m) - \mathrm{sim}_{\mathrm{null}}(n, m)). \tag{3}$$

Here the first term $\mathrm{sim}(n, m)$ is some measure of the similarity of two nodes n and m. The second term, $\mathrm{sim}_{\mathrm{null}}(n, m)$, is the expected value of similarity of these two nodes in some suitable null model. There is a lot of freedom in choosing a null model but in general it is some randomised

version of the DAG. As m and n are in the same antichain $\mathcal{A}$ there is no path between nodes contributing to siblinarity. We have excluded the case $m = n$ so any node in a community by itself contributes zero and $S(\mathfrak{A}) = 0$ for the case where every community is a single node. Including the $m = n$ terms only adds an irrelevant overall constant.

Any similarity measure could be used but a logical choice for the similarity function in our context is the number of neighbours that n and m have in common, so $\text{sim}(n, m) = |\mathcal{N}(n) \cap \mathcal{N}(m)|$ where $\mathcal{N}(n)$ is the neighbourhood of node n. There are two obvious choices for this neighbourhood: one in terms of its predecessors $\mathcal{N}^{(\text{pre})}(n)$, as used in the family tree example above, and another in terms of the successors, $\mathcal{N}^{(\text{suc})}(n)$. More formally

$$\mathcal{N}^{(\text{pre})}(n) = \{m | (m, n) \in \mathcal{E}\}, \qquad \mathcal{N}^{(\text{suc})}(n) = \{m | (n, m) \in \mathcal{E}\}. \tag{4}$$

We could also use both, and use $\mathcal{N}^{(\text{both})}(n) = \mathcal{N}^{(\text{pre})}(n) \cup \mathcal{N}^{(\text{suc})}(n)$.

It is useful to express this in a matrix form as follows

$$S(\mathfrak{A}) \;=\; \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n \in \mathcal{A}} \sum_{m \in \mathcal{A} \backslash n} \left(\tilde{A}_{nm} - \frac{\kappa_n \kappa_m}{W} \right), \tag{5}$$

$$\kappa_n := \sum_m \tilde{A}_{nm}, \quad W = \sum_{n,m} \tilde{A}_{nm}.$$

Here the adjacency matrix $\mathbf{A}$ for our DAG is defined so that A_{nm} is the weight of the edge from n to m. The neighbourhood overlap is captured by the matrix $\tilde{\mathbf{A}}$ which is the product of the adjacency matrix $\mathbf{A}$ of the DAG and its transpose. The $\tilde{\mathbf{A}}$ matrix can be regarded as the adjacency matrix for a derived graph $\tilde{\mathcal{G}}$ which is a directed weighted graph with the same node set as our DAG. In the case where we have an unweighted DAG we define this to be either $\tilde{\mathbf{A}}^{(\text{suc})}$, our successors-based similarity matrix, or $\tilde{\mathbf{A}}^{(\text{pre})}$ is a similarity matrix based on predecessors[2] where

$$\tilde{\mathbf{A}}^{(\text{suc})} = \mathbf{A}.\mathbf{A}^{\text{T}}, \quad \tilde{\mathbf{A}}^{(\text{pre})} = \mathbf{A}^{\text{T}}.\mathbf{A}. \tag{6}$$

The κ_n is the strength of edges attached to a node n in a graph with similarity matrix $\tilde{\mathbf{A}}$ and W is the total strength of edges in that graph. The second term in (5) also defines the null model we use. This is a configuration model applied to the derived graph $\tilde{\mathcal{G}}$.

The form we use (5) emulates the definition of modularity [19], a similarity measure minus expected value of that similarity measure in some null model. The biggest difference between modularity and siblinarity is that we impose the antichain constraint in the communities we study.

This comparison with modularity (see [1] for a review) suggests that we can modify sibl-linarity to adjust the typical number of antichains found, the resolution of our method. One simple method is to scale the null model term [20] so that

$$S(\mathfrak{A}, \lambda) \;=\; \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n \in \mathcal{A}} \sum_{m \in \mathcal{A} \backslash n} \left(\tilde{A}_{nm} - \lambda \frac{\kappa_n \kappa_m}{W} \right). \tag{7}$$

We will show some examples of this in the appendix, see Fig. A3. However it is clear that for large λ we expect many small antichain communities. In particular, for $\lambda \gtrsim W$ adding any node in a community by itself to any other antichain will reduce siblinarity so we expect all the antichains to be the trivial case where each antichain has just one node. Conversely, we expect $\lambda = 0$ to produce large antichain communities.

---

[2]Should we choose to use both sets of neighbours then we simply use the sum of these two matrices $\tilde{\mathbf{A}}^{(\text{both})} = \tilde{\mathbf{A}}^{(\text{suc})} + \tilde{\mathbf{A}}^{(\text{pre})}$.

2.4 Optimisation

We can use any numerical optimisation scheme to find an antichain partition that gives a value for siblinarity $S(\mathfrak{A})$ of (40) which is close to a maximum value for siblinarity. In this paper we choose to adopt the Louvain approach [21] to community detection in networks. We chose this because it is fast and successful at finding communities in networks and because it proved easy to adapt to our context. In this method we first use a greedy optimisation phase. In this we try moving single nodes into a different antichain community, choosing the community structure which gives the biggest siblinarity value, even if that means leaving the structure unchanged, We sweep through all the nodes many times until there are no more changes in siblinarity. We then initiate the second stage in which we produce a new weighted directed network $\mathcal{H}$ in which each antichain community in the original network is now represented by a single node in the new derived network. Edges are merged to match this new vertex set, keeping the total weight of the graph unchanged. Note that this derived network is not necessarily a DAG, see Fig. B4 for an example. However, while our discussion has been framed in terms of a DAG, the concept of an antichain is useful in any directed network with few small cycles and all our expressions, e.g. for siblinarity, remain valid. One then repeats the greedy optimisation with this new derived graph, first a greedy step and second a projection onto a smaller derived graph. One can stop the process with any of the derived graphs but it will terminate when there are no pairs of nodes (antichains in the original graph) which can be merged into a new antichain such that siblinarity is increased.

There are a few key differences from the traditional Louvain approach to modularity optimisation. First, for large DAGs we found it convenient to terminate the first phase on the original data early. That is we stopped when we had some non-trivial antichains in our partition but before we had exhausted all possible single node moves which increased siblinarity. It is not clear if this results in antichain partitions at the end of the process are better or worse in terms of the siblinarity value but it results in a significant speed increase as working on the smaller derived networks is always faster. In a similar way, one can terminate the iterative scheme at any point where the communities seem useful, if one can identify what 'useful' means. A large value of siblinarity is only one measure of usefulness of our antichain partitions.

2.5 Antichain Measures

Once we have found our antichains, we will need to analyse their properties and we will need some additional tools to do this for large examples.

An interesting question to ask is whether nodes in an antichain are similar to other nodes in the antichain. Many traditional methods for measuring the strength of a community are based on the number of edges between members of the community compared with edges to those outside. However such measures fail for our antichain communities where there are no edges between community members. However, in constructing siblinarity, we have used a similarity measure based on the number of common neighbours in a suitable set of neighbours $\mathcal{N}(n)$, such as the set of successors $\mathcal{N}^{(\mathrm{suc})}(n)$ or predecessors $\mathcal{N}^{(\mathrm{pre})}(n)$. Any similarity measure could be used in principle.

So for each antichain community $\mathcal{A}$ we define a similarity matrix $\mathbf{W}(\mathcal{A})$ where

$$W_{nm}(\mathcal{A}) = |\mathcal{N}(n) \cap \mathcal{N}(m)|, \qquad n, m \in \mathcal{A} \tag{8}$$

is equal to the number of common neighbours of nodes m and n. There are many ways to use

this. We can then use multiple metrics to give insights about how interconnected $\mathcal{A}$ is. For instance, we can define $W(\mathcal{A})$ to be the average weight per node in an antichain $\mathcal{A}$,

$$\frac{W(\mathcal{A})}{|\mathcal{A}|} = \frac{1}{2|\mathcal{A}|} \sum_{n,m \in \mathcal{A}} W_{nm} \tag{9}$$

reflects on the mean similarity of nodes in the antichain. If this value is small, and the siblinarity of a given antichain,

$$S(\mathcal{A}, \lambda) = \sum_{n \in \mathcal{A}} \sum_{m \in \mathcal{A} \setminus n} \left(\tilde{A}_{nm} - \lambda \frac{\kappa_n \kappa_m}{W} \right) \tag{10}$$

is large, we can expect that the overlap of neighbourhoods of nodes in the antichain is sparse. In Section 4.4 we will use these metrics as well as summary statistics to study antichains, obtained in `hep-th` citation network.

3 Data and Models

We used a variety of models and data sets which have natural representations as a DAG in order to compare different community structures found by different methods.

3.1 Space-Time Lattice Model

Our first test model is a simple DAG where the nodes are placed on a square lattice at (t, x) where tL and xL are integers between 0 and $(L-1)$. To place edges between the node at the points of our lattice, we use the Manhattan distance $d(n, m) = |t_n - t_m| + |x_n - x_m|$ for the distance between nodes n and m of coordinates (t_n, x_n) and (t_m, x_m) respectively. We add a directed edge from n to m with probability $p(n, m)$ where

$$p(n, m) = \begin{cases} 0 & \text{if } t_n \geq t_m \\ 1 - \frac{d(n,m)}{D} & \text{if } t_n < t_m. \end{cases} \tag{11}$$

The edges from n to m only exist if $t_n < t_m$ and it is this arrow-of-time which ensures that we will always have a DAG.

The idea behind this DAG model is that nodes there is a natural hierarchy given by the t-coordinate which guarantees acyclicity. In addition, nodes which are close in their space and time coordinates will often have large numbers of common neighbours, both successors and predecessors, whereas nodes that are distant in space or time will have few neighbours in common. So in the visualisations we expect to see nodes of the same time coordinate and close in their space coordinates, to be placed in the same antichain community. However, as the links are placed with a stochastic mechanism, we will sometimes see nodes from neighbouring layers grouped together. That is the antichain structure in a given realisation of our model is not a perfect match to the natural layers defined by the time coordinate. This simulates what one finds in real data sets. For instance, two papers written on a similar topic would be represented by nodes with a similar x coordinate in this model. If they were written independently at the same time then they could not be connected but if published at slightly different times it might be possible for the earlier paper to be cited by the later paper.

8

3.2 Price Model with Subject Fields

One of the oldest models of citation networks is the Price model of cumulative advantage [22] and this defines a DAG which has the fat-tailed (power-law) distribution for the number of papers with a given citation count[3]. Here we consider a modified version of the Price model in which we assign nodes (representing papers) to different 'fields' and we create edges (citations between papers) such they are usually between papers in the same field. This model is too simple to capture many aspects of real citation network though it does emulate three fundamental aspects: the order of papers imposed by time, the fat-tailed citation count distribution, and the preference of most papers to cite papers within a similar field. For our purposes, all we use here is that this gives a DAG with a well known structure and now with a planted partition. Our expectation is that our siblinarity antichain partitions will tend to cluster papers published around the same time and in the same field.

The model is illustrated in Fig. 2. To define this model, consider a sequence of networks $\mathcal{G}(t)$ where t is a positive integer playing the role of time and which gives us an order to the nodes in our networks. Each graph $\mathcal{G}(t)$ has t nodes. In our notation, the node $u(s)$ is always the node added at step s in the process so it exists in all networks $\mathcal{G}(t)$ provided $0 < s \leq t$. The nodes in these networks are also partitioned into different fields, that is each node $u(t)$ is in field $f(t)$, one of F distinct fields.

To create the next graph in the sequence, $\mathcal{G}(t + 1)$, we first add a new node $v(t + 1)$ to the vertex set. This new node also is assigned to a field, $f(t+1)$, chosen uniformly at random from the set of F possible fields.

We now add m directed edges to this new node $v(t + 1)$ from existing nodes $u(s)$ where $s \leq t$. Following Price, we chose the source nodes s for the new edges from the existing network $\mathcal{G}(t)$ with probability $\Pi(t, s)$ defined to encode "cumulative advantage". That is the higher the current citation count of a paper, the more likely it is to be cited. Since we define our directed edges to run from early to late times, this means our probability $\Pi(t, s)$ is proportional to a linear function of the current out degree of existing nodes, say $k^{\text{out}}(t, s)$ for the citation count of node s in the graph $\mathcal{G}(t)$. For simplicity we choose Price's original form $\Pi(t, s) \propto (k^{\text{out}}(t, s)+1)$.

To impose a modular structure reflecting the preference of appears to cite others papers within the same field, we use a parameter ϕ which is the probability that a new paper $v(t + 1)$ cites another paper $u(s)$ in its own field i.e. $f(s) = f(t)$ but where the connection is made to node $u(s)$ chosen from the papers in the same field using our cumulative advantage $\Pi(t, s)$. With probability $(1 - \phi)$, the source node $u(s)$ of a new edge is chosen from within the papers not in the same field i.e. $f(t + 1) \neq f(s)$.

This leaves us with a stochastic model of three parameters: m, ϕ and the total number of nodes in any network we use. We considered networks with $5,000$ nodes and $m = 3$ edges added to each new node. Some aspects of the model can depend on the initial graph used to start the simulation but this was unimportant for our studies. The model is described in more detail in Appendix F.

3.3 Real World Data Sets

To test our approach on actual data, we use three examples.

First we use the Florida Bay food-web dataset [23]. In this network, the nodes are compartments and edges represent directed carbon exchange in the Florida Bay. There is an edge

---

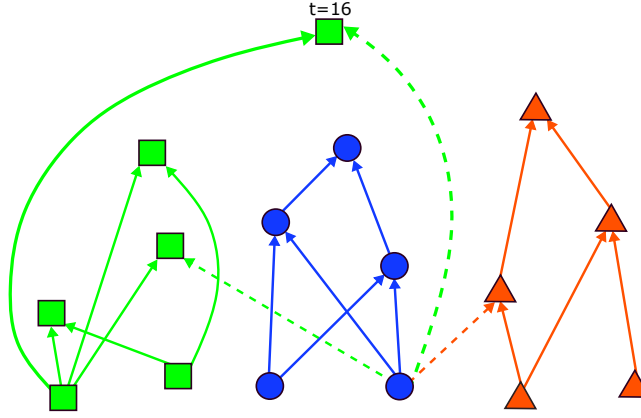[3]The undirected version of this is the Barabási-Albert model, see [18] for a discussion.

Figure 2: An illustration of the Price model with two edges added per node ($m = 2$) and three fields ($F = 3$) as indicated by the three types of node. Time increases as you move up the diagram so the vertical ordering of the nodes represents a total order in the DAG. Solid (dashed) lines represent citations between papers of identical (different) fields. A new node labelled $16 = (t + 1)$ in field 0 (green squares) is added an existing network of fifteen nodes. Here we suggest the two edges added to node 16 are to the two nodes of highest degree (the cumulative advantage bias) with one in the same field (continuous edge) and one in a different field (dashed edge).

from i to j if compartment j consumes carbon from compartment i (often, this means species j eats species i). The compartments are mostly organisms but also encompass special nodes such as "input" and "output". We also had group classification labels. Examples groups include "Zooplankton Microfauna" and "Pelagic Fishes" [24]. The original network consists of 128 nodes and 2106 edges but this contains cycles. We used the breadth first search approach, described in [25] to recover a DAG, removing 176 edges (less than 9%). .

We also used two examples of a citation networks in which documents are nodes and we draw an edge from a newer document when it cites an older document. We used two datasets: citations between papers in the arXiv High Energy Physics-Theory repository, for papers published between 1992 and 2003 (referred to here as `hep-th`) [26] and Cora [27][28]papers, referred below as `cora`. The `hep-th` dataset contains a set of 27,770 papers and 351,500 edges, that represent citations between the papers. The `cora` dataset consists of 2,708 scientific publications classified into one of seven classes, based on their topics. The citation network consists of 5,429 links[4]. We use a convention that a citation of a paper u by a paper v is represented by an edge (u, v).

---

[4]In practice there are often "bad" links which are in the "wrong" direction, from a newer document to an older document. This is because documents are published in different versions and the text available may not have been created at the time associated with the document in the data set. For instance, a revised version of an arXiv paper carry the same index as the first version. A journal article has several associated dates: first submitted, date accepted, published online, formal publication date and so forth. Such bad links can introduce cycles and these must be dealt with. For our arXiv data we simply dropped the links that do not respect the agreed order of time; they account for less than 1% of the data. For the `cora` data we used the approach, discussed in [25] to recover a DAG from a directed network, leaving us with a network, composed of 5,255 edges.

4 Results

4.1 Space-Time Lattice

We use the space-time lattice model of section 3.1 on a small scale because we can make effective visualisations which illustrate the key ideas of communities in DAGs. In Fig. 3 we show the same instance of the space-time lattice model in which nodes are placed on an eight-by-eight square lattice. We then use colour to show the different partitions produced using different algorithms.

The bottom two examples of Fig. 3 illustrate standard community detection algorithms applied to our network. Here these are found by optimising modularity with two different resolutions (obtained by scaling the null model term). The communities found tend to be located in one region of space reflecting the spatial constraint in the model. However, they extend over several values of time so these communities do not respect the order inherent in the DAG. That is, the nodes in the these Louvain communities are not similar in terms of their time coordinates.

If this was a citation network, we would be grouping papers together with different publication dates. If we were in interested in comparing the impact of each paper through citation counts, the older papers would have an advantage making the comparison unfair. If this toy model represented a food web, these communities derived from the undirected network might capture aspects such as distinct parts of the ecosystem. However it fails to find the hierarchy, each group would contain predators and their prey.

So these communities from the undirected graph may be of use in some contexts, but they are not particularly sensitive to the inherent structure of the DAG coming from the time direction.

The middle two examples in Fig. 3 use the height- and depth-antichain partitions. Now the hierarchical structure coming from the time direction is clearly exposed, with many communities running across one row, perhaps two. Where two rows are involved, it is showing how the these communities detection methods are highlighting where the placement of the nodes in the visualisation does not reflect the topological reality because of the stochastic aspect of edge placement. This shows that these antichain partitions can do a useful job, picking out the true topological based hierarchy as defined by the data.

However the height- and depth-antichain communities show the opposite problem from the undirected graph communities. That is they respect the order coming from the arrow of time but they fail to pick up in any way the spatial clustering in the data. If this was a citation network, these would cluster papers published at a similar time regardless of academic field. For a food web, all predators at the same level would be grouped together regardless of whether or not they were competing in the same ecological niche.

The top two networks in Fig. 3 show the siblinarity antichain communities, one based on predecessor neighbours and the other using successors. This shows that the communities respect both the time and the spatial clustering in the data. The nodes tend to be in the same row and only contain nodes which are close by reflecting the spatial clustering in the model. This suggests that when applied to a citation network, this method will cluster papers which are directly comparable, similar publication date and similar field. Applied to food webs, siblinarity will only group predators at the same level competing in the same niche.
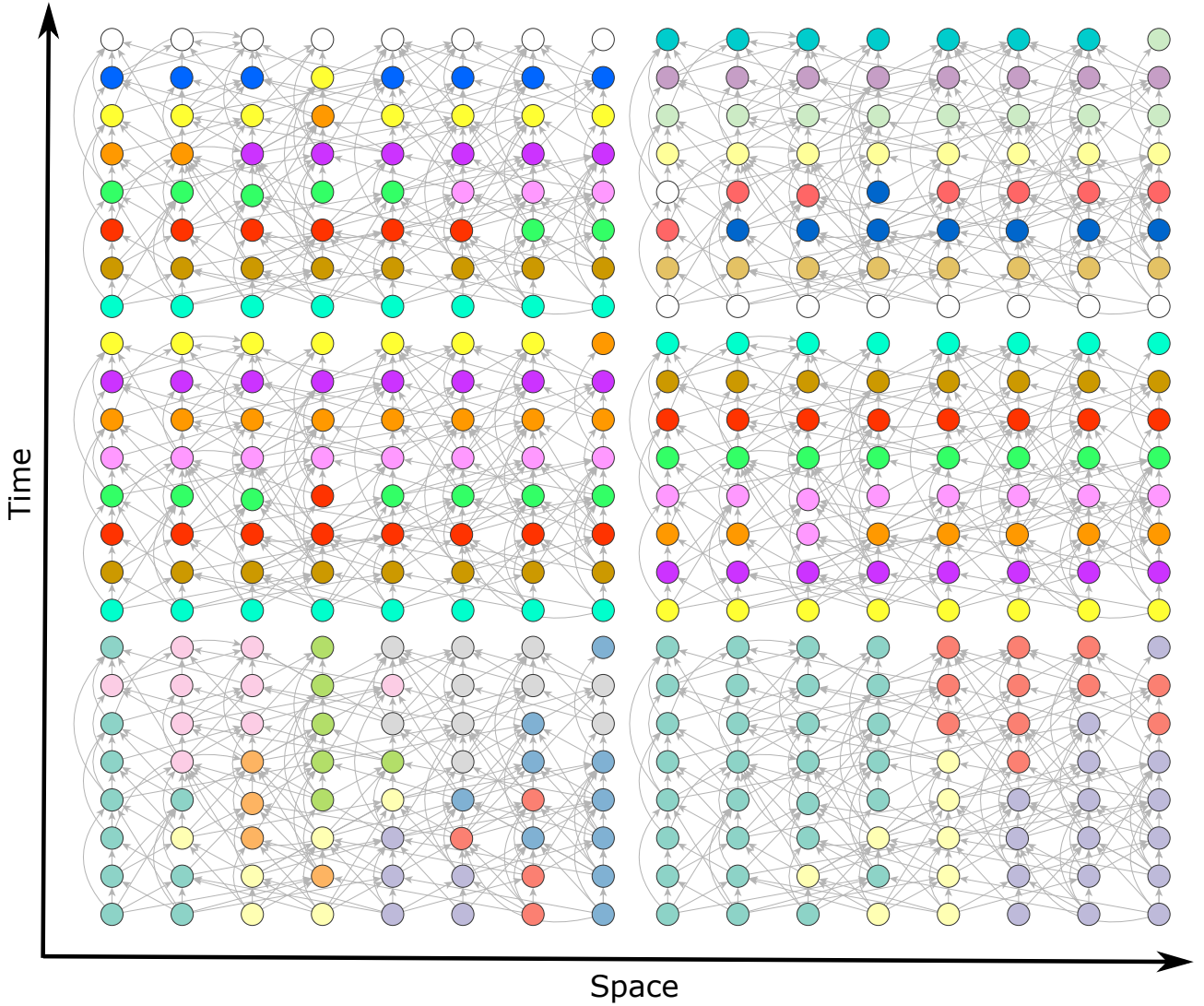
Figure 3: Examples of various partitions of DAG from a simple space-time model. In this DAG, edges are more probable between two nodes if they are a shorter Manhattan distance apart (in the space-time). The time coordinate, which induces the acyclicity in the network, is vertical. Colour indicates community of a node. The top row shows siblinarity communities based on common successors (A), and predecessors (B). The central row shows layers based on height (C) and depth (D). The communities in the bottom row are from Louvain community detection using resolution parameter value of one (E) and two (F). Nodes coloured white are in a community of size one. Siblinarity partitioning tends not to form communities (top row) which stretch across the whole network unlike those communities based on height or depth shown in the centre row. The communities found using traditional methods respect the spatial (horizontal) constraint but show no respect for the order in the DAG as they are spread over several times vertically.

4.2 Diversity of Antichains: The Price Model with Fields

The space-time model illustrated that the communities formed by our methods respect the order implicit in a DAG. Our method also aims to create groups which are similar in other ways as reflected in the network structure. To test this we use our modified version of the Price

model of section 3.2. We will use the language of a citation model, reflecting Price's original context. So here we say we are aiming to group papers (nodes) of a similar age (in an antichain) published in the same academic field based on the network structure alone (using co-citation or bibliometric coupling).

Here we consider the case when a field is assigned stochastically using a uniform distribution so that it is equally likely to assign a field $f = 1$ and $f = 5$. Other variations are possible, for instance, one could assign the fields sequentially deterministically or create non-uniform in size fields. Some of these variations will be discussed in Appendix F.

Our aim is to show that our antichain partitions are largely composed of papers from the same field which we measure using Shannon's diversity metric D [29]. That is given an antichain $\mathcal{A}$ we find p_f, the fraction of nodes in the antichain which are in field f. The diversity of this antichain is then given by

$$D(\mathcal{A}) = \exp\left(- \sum_f p_f \ln(p_f) \right) \tag{12}$$

so that $1 \leq D(\mathcal{A}) \leq |\mathcal{F}|$. We also define the AGE of an antichain as the mean age of the nodes within it:

$$a_{\mathcal{A}} = \frac{1}{|\mathcal{A}|} \sum_{n \in \mathcal{A}} t_n \tag{13}$$

where t_n is the 'age' of a vertex n, i.e. node n was added at step t_n.

The results for the average diversity of a partition, $\sum_{\mathcal{A} \in \mathfrak{A}} D(\mathcal{A})/|\mathfrak{A}|$ are shown in Fig. 4, results for the distribution of $D(\mathcal{A})$ values within $\mathfrak{A}$ are shown in Fig. 5. Not surprisingly the diversity of height and depth antichains is almost always larger than that of communities found from a siblinarity antichain partition. This is expected because the height and depth antichains typically contain more nodes, reflected by the low number of points on their plots, as their construction is not effected by the field labels. Since they are constructed without regard to the field labels, we would expect, and find, that the diversity of the height and depth partitions is close to the maximum value where $p_f \approx 1/|\mathcal{F}|$ and so $D \approx |\mathcal{F}|$.

We also noted that the diversity of antichains that are height or depth based does not approach maximum if the sizes of fields are not uniform. This is expected as Shannon's entropy not only accounts for number of species in the ecosystem but also their relative abundances. Uneven sizes of fields result in expected diversity score lower than maximum, as the maximum would be obtained if we had equally abundant species, as seen in the other two figures. See Appendix for the case where field sizes vary.
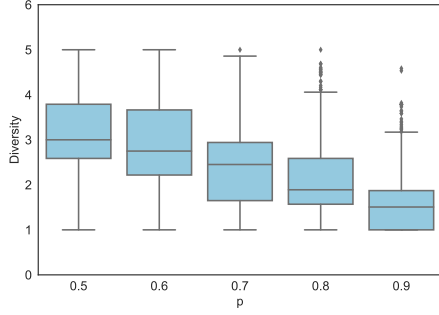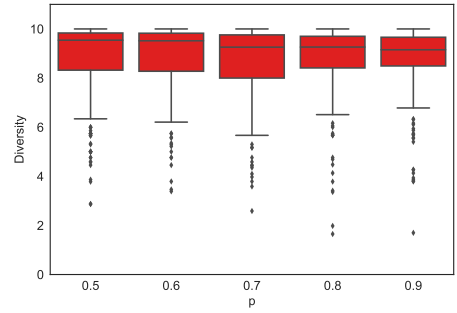
(a) Diversity of successor siblinarity antichains in a five-field Price model.
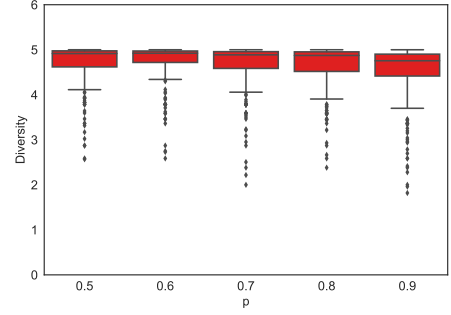
(b) Diversity of height antichains in a five-field Price model.
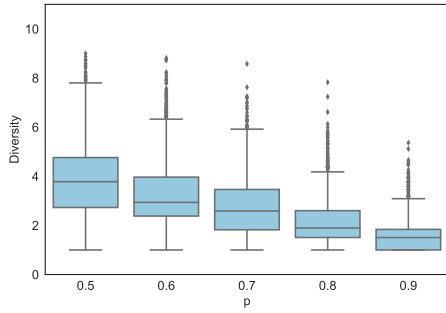
(c) Diversity of successor siblinarity antichains in a ten-field Price model.

(d) Diversity of height antichains in a ten-field Price model.

Figure 4: Comparison of diversity in height-based antichain partitions (red boxes on right) versus siblinarity communities, based on common successors (blue boxes on left) in the Price model with five fields (top row, Fig. 4b and Fig. 4b) and ten fields (bottom row, Fig. 4c and Fig. 4d) and different intraconnectivity probability p. We studied ten networks of $5,000$ nodes for each set of parameters. Figures show that the diversity of siblinarity antichains is clearly significantly smaller than that of heights. The box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data. Flier points are those past the end of the whiskers.
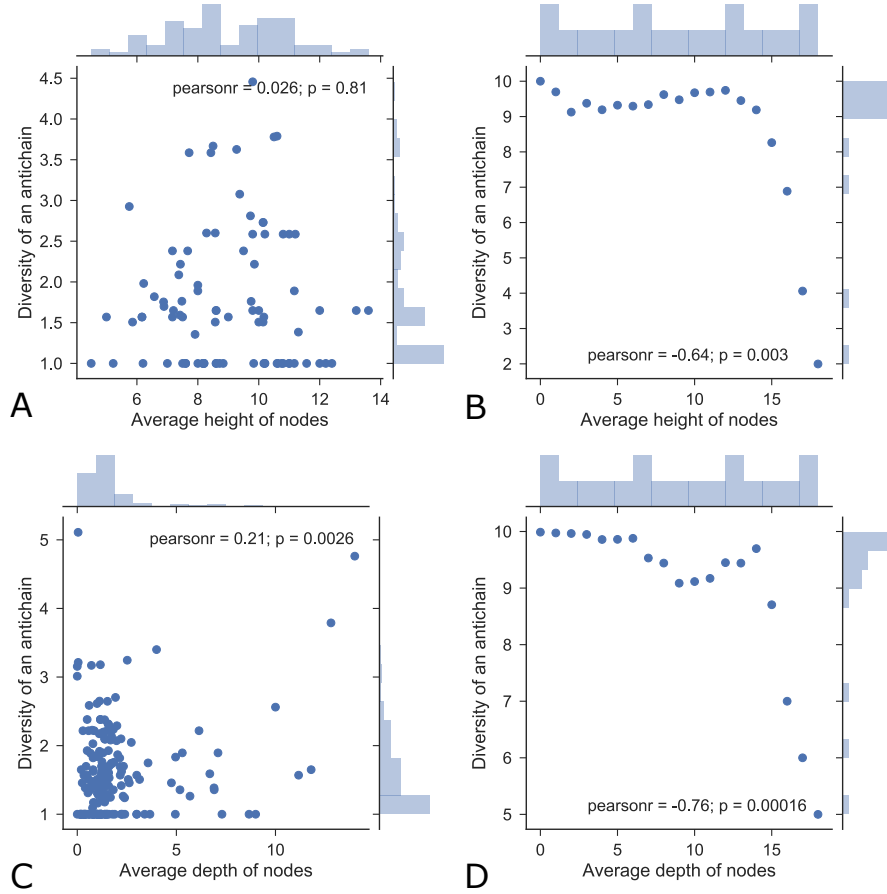
Figure 5: Shannon's diversity of antichains in a Price model with planned partition of nodes into five fields for networks of $5,000$ nodes. Each node is assigned to a field sequentially and nine times out of ten a new node chooses to connect to an existing node of the same field, with a preference for large degree nodes, $\phi = 0.9$. Each new node attaches to $m = 3$ older nodes. Each point represents an antichain in a given partition, with the average age $a_\mathcal{A}$ (13) and Diversity $D(\mathcal{A})$ (12) used as coordinates. Four different types of antichain partition are shown: A - successors-based siblinarity antichains, B - height-based antichains, C - predecessors-based siblinarity antichains, D - depth-based antichains. The height and depth antichains are clearly much bigger (few points on their plots) but much less diverse, with their diversity close to the theoretical maximum 10.0. On the other hand, the siblinarity antichains based on co-citation similarity (A) or bibliometric coupling (B) are drawn largely from the same field with diversities close to the minimum value of 1.0.

4.3   Florida Bay food web

We also studied our antichain partitions in the DAG version of the Florida Bay food web data set [23]. Results are shown in Table 1.

No.	Species (type)
1	2μm Spherical Phyplankt, Synedococcus, Oscillatoria, Small Diatoms ($< 20\mu m$), Big Diatoms($> 20\mu m$), Dinoflagellates, Other Phytoplankton, Benthic Phytoplankton, Thalassia, Halodule, Syringodium, Roots, Drift Algae, Epiphytes.
3	Acartia Tonsa, Oithona nana, Paracalanus, Other Copepoda, Other Zooplankton, Sponges, Bivalves.
7	Coral, Other Cnidaridae.
9	Benthic Crustaceans, Detritivorous Amphipods, Herbivorous Amphipods, Detritivorous Gastropods, Detritivorous Polychaetes, Suspension Feeding Polych, Macrobenthos, Detritivorous Crabs.
18	Toadfish, Brotalus.
19	Other Killifish, Goldspotted killifish, Blennies, Clown Goby, Silverside, Lobster, Predatory Crabs, Callinectus sapidus, Bay Anchovy, Rainwater killifish, Mullet, Other Horsefish, Gulf Pipefish, Dwarf Seahorse, Code Goby, Halfbeaks.
22	Flatfish, Grunt, Pinfish, Rays, Porgy, Scianids, Parrotfish, Bonefish, Needlefish, Snook, Puffer, Manatee.
23	Omnivorous Crabs, Pink Shrimp.
25	DOC, Isopods, Herbivorous Shrimp, Thor Floridanus, Sailfin Molly, Green Turtle.
26	Sharks, Tarpon, Lizardfish, Grouper, Jacks, Pompano, Gray Snapper, Red Drum, Mackerel, Small Herons & Egrets, Ibis, Roseate Spoonbill, Herbivorous Ducks, Omnivorous Ducks, Gruiformes, Small Shorebirds, Gulls & Terns, Kingfisher, Loggerhead Turtle, Hawksbill Turtle.
27	Other Snapper, Other Pelagic Fishes, Spotted Seatrout.
28	Stone Crab, Sardines, Anchovy, Other Demersal Fishes, Filefishes.
29	Barracuda, Loon, Greeb, Pelican, Comorant, Big Herons & Egrets, Predatory Ducks.
33	Free Bacteria, Dolphin.
35	Raptors, Crocodiles.
36	Output, Respiration.
Ind.	Input (0), Water Flagellates (2), Water Cilitaes (4), Benthic Ciliates (5), Meroplankton (6), Meiofauna (8), Benthic Flagellates (10), Water POC (12), Predatory Gastropods (13), Echinoderma (14), Predatory Shrimp (15), Predatory Polychaetes (16), Mojarra (20), Benthic POC (24), Spadefish (31), Catfish (32), Eels (34).

Table 1: Siblinarity communities, based on common prey and common predators in the Florida Bay food web. Some communities consist of a single species and these are listed together under community number "Ind." to reduce the size of the table. The numbers in brackets indicate the community index in the induced graph, shown in Fig. 6. Note the green turtle in community 25 appears to be out of place with the other smaller species in the same community but they all tend to feed on similar species.

We see that many of our siblinarity antichains consist of similar species as indicated by their similar names. For instance community 9 consists of, amongst other crustaceans, several types of amphipods. Another example of how our siblinarity antichains work is the example of the green turtle in community 15. The "Green Turtle" node has been placed in a community along with much smaller species, seemingly very different from turtles, such as shrimp and isopods. However, green turtles feed on thalassia (a type of seagrass, commonly known as turtlegrass), which is also a food source for organisms, represented with "DOC" (Dissolved Organic Carbon) node and isopods. Furthermore, all of the species in this green turtle antichain community feed on epiphytes. Nodes in this antichain all have the same height of two, but their depths range from 22 for "Epiphytic Gastropods" to 30 for "Isopods", "Herbivorous Shrimp", and "Thor Floridanus". Another example of an antichain, in which nodes of a variety of heights and

depths were collected into a siblinarity community is the community 27, consisting of "Other Snapper" (height 27, depth 4), "Other Pelagic Fishes" (height 28, depth 4) and "Spotted Seatrout" (height 29, depth 3).

In Fig. 6 we represent the siblinarity communities as nodes in the induced graph $\mathcal{H}$ (see section C). Two communities are connected if at least one species in one antichain preys on at least one species in the other. The communities are placed vertically based on the average height of the nodes in each community so we can see many examples where species at the same height, the trophic level, are broken up into smaller groups. For instance nodes with height 28 are split in between three siblinarity communities, with indices 22, 26 and 28.
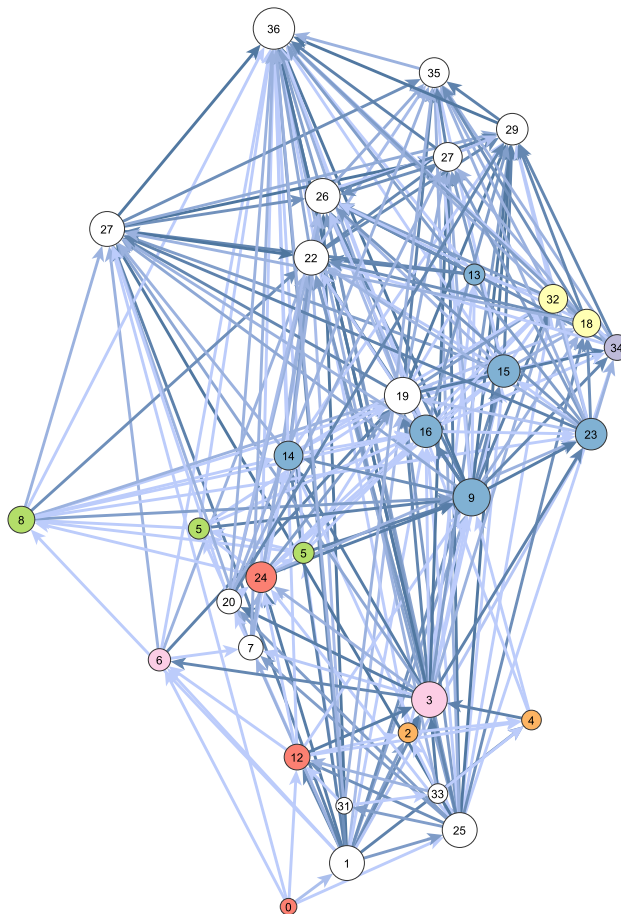

Figure 6: Siblinarity communities in a Florida Bay food web, based on common successors (prey). The figure shows the induced graph $\mathcal{H}$ (see section C), in which each node here represents one siblinarity community in the Food web DAG. An edge from one community to a second shows that the first community contains at least one species which is the prey for at least one species in the second antichain. Node colours represent the most common type of species (white means it is unspecified), size represents the size of an antichain, the vertical position approximately corresponds to the height of nodes in the induced graph. One can see that there are often multiple siblinarity antichains per height (or across similar heights), that there are a wide range of sizes of antichains, and the nodes that are grouped into antichains are qualitatively similar.

17

4.4 Citation networks

The Science Citation Index, introduced by Garfield in 1964, has a functionality to search for "Related Records" [18]. A Related Record is any record which shares at least one cited reference with the original source record. The more shared references, the more closely related the records are — an extension of the notion of citation searching to track a subject area. The related records are further ranked based on the number of shared references.

So in a bibliometric context our approach is an extension for "Related Records" functionality in the SCI. As pointed out by Newman, simply using bibliographic coupling or co-citations is flawed: strong bibliographic coupling or co-citations only occur between papers that have either large bibliographies or are highly cited, respectively [18]. Our approach naturally eliminates this flaw: by comparing the observed neighbourhood overlap to a null model, we can evaluate the significance of the similarity regardless of the degrees of two nodes.

To understand what type of nodes are joined together in antichains and why, we looked at various statistics for successor siblinarity antichains in the `hep-th` dataset. These neighbours are newer papers that cite the papers in the antichain communities. We found 10,808 successor based antichain communities, 7,352 of which are composed of singular nodes and 1,226 are composed of at least five nodes. Several of the measures we found useful can be defined in terms of a bipartite network constructed for each antichain community. This is simply the undirected subgraph of the full network, containing the nodes in the antichain of interest, the nodes of all their neighbours (here successor neighbours) and then all the edges between these nodes in the original graph. This is a bipartite network given the two types of nodes (see Appendix D for a formal definition). To remove noise from weakly connected nodes in antichain communities, we also found it was useful to reduce the bipartite subgraphs by dropping any neighbour node that is connected to just one node in the antichain.

We then applied a number of simple network statistics to these bipartite subgraphs in order to understand the nature of our antichain communities. Our results are summarised in Fig. 7. Several simple network statistics, formally defined in Appendix D, proved useful. The most basic measures are the size of each antichain size $|\mathcal{A}|$, and $|\mathcal{N}|$ the total number of neighbours of each community. The ratio of these gives us the average number of neighbours of an antichain node in each community, $\langle k \rangle$, and we also use the standard deviation of that measure, $\sigma(k)$. A low $\langle k \rangle$ indicates a sparse "zig-zag" pattern with very weak overlaps between members of the antichain. A large $\sigma(k)$ indicates that we may have one well connected node in the antichain plus many connected to only a couple of neighbours. The last network statistic shown in Fig. 7 is the density of the bipartite graph, the total number of edges divided by the maximum number possible which is equal to $\langle k \rangle / |\mathcal{N}|$. This final network measure is close to one only if almost every member of our antichain community has the same set of neighbours.

The majority of the successor antichain communities we find in `hep-th` are small, composed of around 10 to 20 nodes. They typically share around 40 neighbours though a few antichain communities can have closer to a 1000 shared neighbours. The distribution of $\langle k \rangle$ peaks around 6 but there is often a sizeable variation in the degree as shown by similarly large values of $\sigma(k)$.

The density of the bipartite graphs, the ratio between $\langle k \rangle$ and $|\mathcal{N}|$ in Fig. 7, shows that the antichain communities in `hep-th` possess a wide range of values. The distribution of the density peaks at values close to $0.25 - 0.5$ for many antichains, but the number of antichains with minimal and maximal values of the ratio is non-negligible.

Our last statistic is not a network measure but exploits information we have on the month of publication for each paper in our `hep-th` dataset. We looked at the standard deviation in

the ages of papers in each antichain. We see that nodes that feature in the same antichain are also nodes of similar age, which varies by less than one year for most antichains.

We also looked at siblinarity values obtained by antichains in `hep-th` and how they relate to $W(\mathcal{A})/|\mathcal{A}|$, the mean similarity of nodes in an antichain as defined in 2.5, as well as the size of antichains $|\mathcal{A}|$. Fig. 8 shows that for a given size of an antichain community, there are a wide range of values for the average neighbour overlap $W(\mathcal{A})/|\mathcal{A}|$ and the siblinarity measure. Higher overlap tends to produce a better quality antichain, that is a higher siblinarity value, but there is still a large variation, reflecting the role of the null model.

For instance some large antichains have small siblinarity scores. Conversely, the largest siblinarity score is obtained from an antichain community which is composed of 68 nodes (antichain with an index 2172 in our data), large but not the largest we found. This antichain is composed of papers, published in 1992–1994, some of which are relatively highly cited (large out-degree). The paper with the largest out-degree of those in this community has an out-degree of 112 (paper index 9201061).

We also studied a second citation network, the `cora` dataset, in which each node is labelled by one of seven different topics. We use this label to quantify the diversity of the nodes in the antichain communities we find, similar to the analysis in Section 4.2. Fig. 9 shows that the diversity of 134 siblinarity antichains is much smaller than the partitions based on heights. Furthermore, most of the time the diversity values for siblinarity antichains are equal to one, which indicates that all nodes in the antichain have the same label. This result confirms that the siblinarity communities are communities of similar papers, not just ones published around the same time.
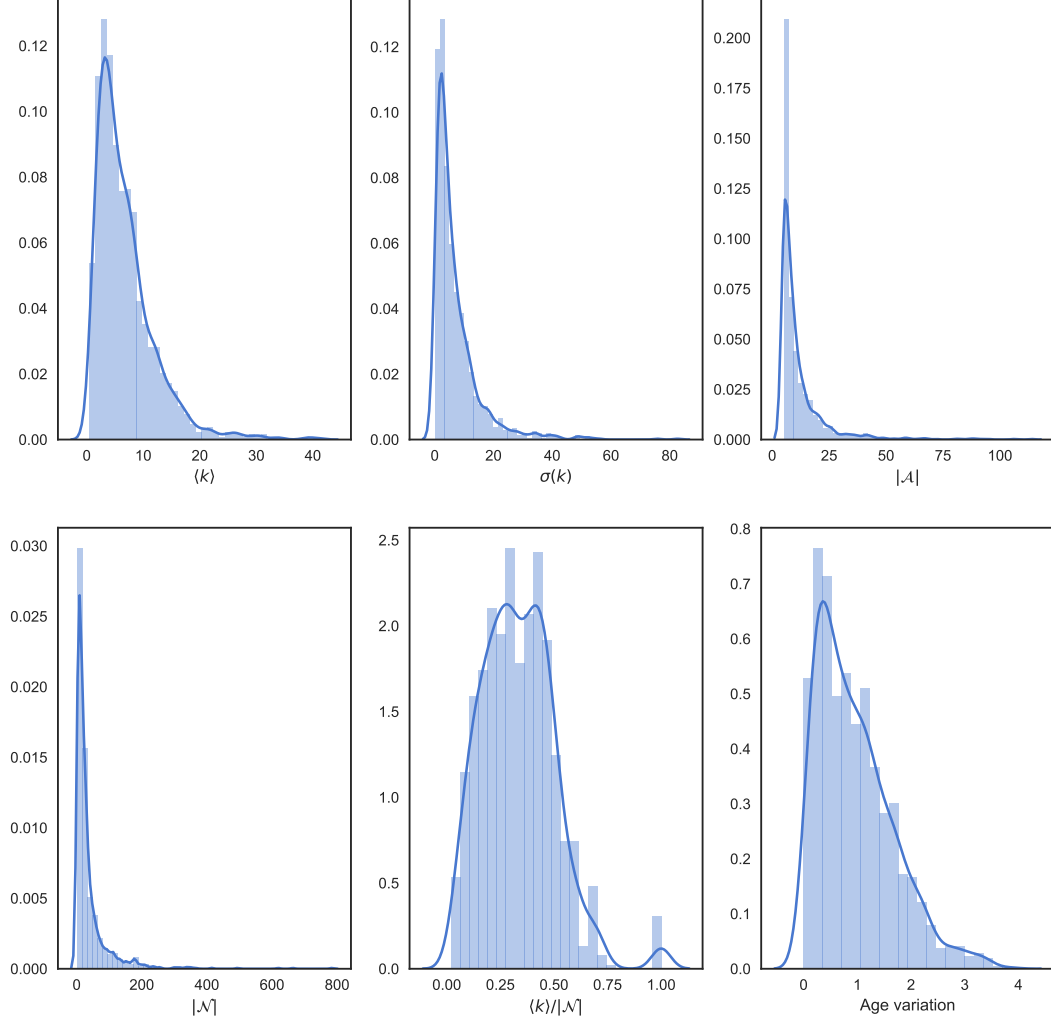
Figure 7: Summary statistics of antichains and shared successors of those antichains, based on which they were obtained in `hep-th` citation network. In each case the relative frequency distribution is plotted over all the antichain communities found using successor neighbours, older neighbours who are cited by the newer papers in the antichain community. The statistics used are as follows: the average number of neighbours of each antichain node $\langle k \rangle$, the standard deviation of the number of neighbours of each antichain node $\sigma(k)$, the size of each antichain community $|\mathcal{A}|$, the density of each bipartite graph $\langle k \rangle / |\mathcal{N}|$. The final panel shows the age variation of papers in each antichain. That is the standard deviation in the publication date of papers in each community, known in terms of months but we give the plot in terms of years.
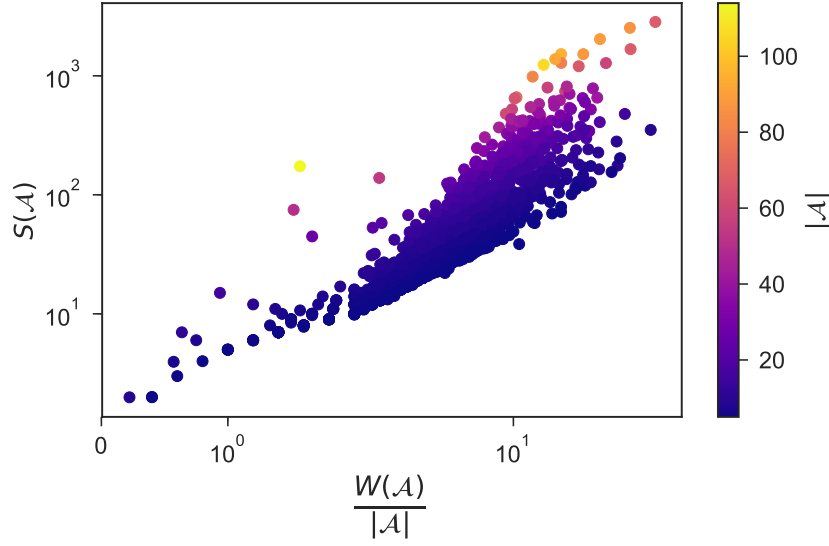
Figure 8: A colour plot showing a relation between siblinarity score of an antichain $\mathcal{A}$, $S(\mathcal{A})$, $W(\mathcal{A})/|\mathcal{A}|$ and $|\mathcal{A}|$ for an antichain partition obtained in `hep-th` citation network. Antichains were obtained considering co-citations. Figure shows that siblinarity scores, obtained by antichains relate to $W(\mathcal{A})/|\mathcal{A}|$ but not to the size of an antichain.
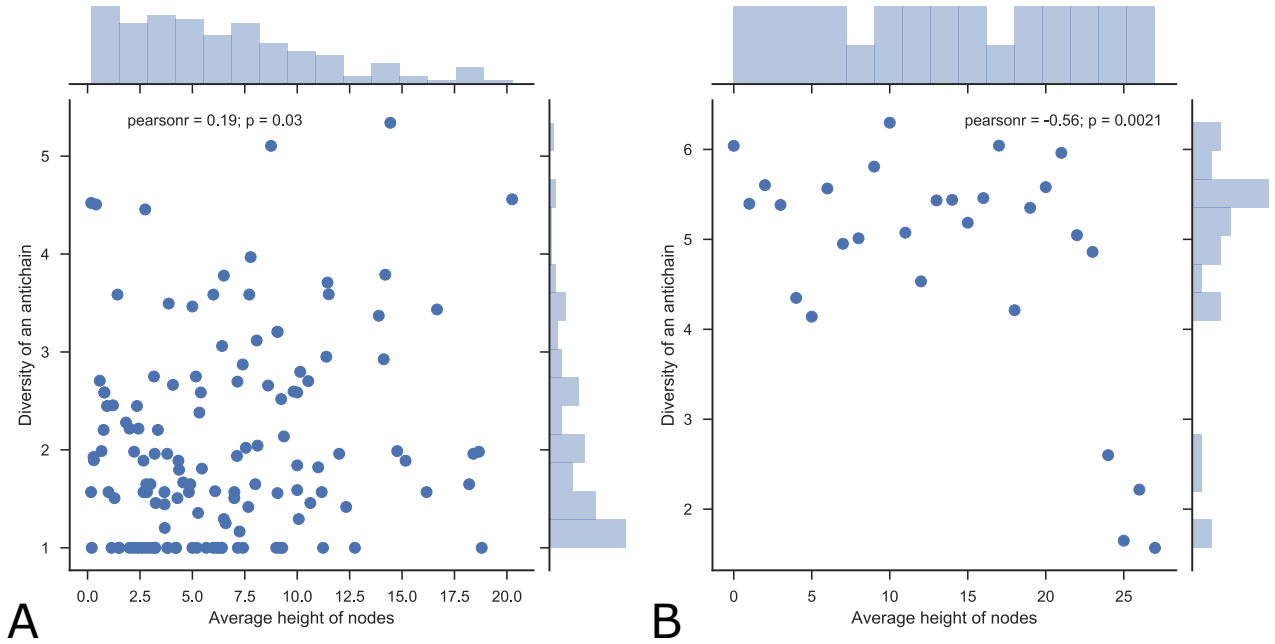

Figure 9: Shannon's diversity of nodes in antichains of the `cora` citation network. Each node has a label assigned to it, based on the topic the paper is on. There are seven topic in total. Two different antichain partitions are shown: A — successors and predecessors-based siblinarity antichains, B — height-based antichains (diversity calculated only for antichains, that are composed of at least five nodes). The horizontal axis is average heights of nodes in the antichain. The height antichains, with diverity values often close to the maximum value of seven, are much more diverse than the siblinarity comminties.

5 Discussion

In network analysis an edge between nodes generally indicates a close relation and a similarity between the two connected nodes. This is exploited to understand local features, such as centrality of individual nodes, as well as gain insights on a macroscopic structure, such as the extent of community structure).

In this work we have highlighted that in some circumstances, the absence of a pairwise relationship can be a just as useful a signal as the presence of it. Our focus has been on Directed Acyclic Graphs (DAGs) where the implicit order in such networks prohibits the direct connection of many similar nodes. As this order is intrinsic to the very nature of a DAG, our response has been to embrace this order as it reflects important features of the data. To find meaningful communities, we have turned to antichains, sets of disconnected nodes which often play a useful role in the study of DAGs and which reflect the topological properties on the networks we consider. To ensure our antichains contain similar nodes, we have used neighbourhood overlap to define the siblinarity of a partition of a DAG into antichain communities.

We have shown using many examples how the antichains which maximise siblinarity are interesting and relevant. Our communities are extremely different from usual communities found in network analysis as the simple examples in Fig. 3 show. The communities in traditional network analysis consist of strongly interconnected nodes while siblinarity communities consist of nodes that are strictly not connected, but which share similar neighbours.

This novel type of node partitioning has several potential uses whenever data is well represented by a DAG. In a citation network, each antichain community would be composed of publications that are similar thematically and which were published at a similar time. So this approach could be used as a recommender system for publication search: given an input paper, one could find alternative similar publications by looking into the antichain of the input paper. Another common practice in bibliometrics is partitioning a citation network into subsets of nodes published in the same year in order to make comparisons between those publications. This is to avoid the bias that the arrow-of-time introduces, for instance older papers have more citations on average than younger papers. However, such clustering by publication date is very similar to clustering by height and this means papers from completely different fields are in the same community. Our method produces clusters of similar age and topic.

Our siblinarity algorithm, like any clustering algorithm, can also be used to coarse-grain a DAG which can help navigate a large amount of data. In our case we would contend the order of a DAG is important. Having a predator and its prey in the same community may not be a useful classification when examining food webs. Indeed we exploit this coarse graining in our implementation of the Louvain algorithm to find antichain partitions which have approximate maximal values of siblinarity.

More broadly, we see our work as emphasising that the order found in some data is an important feature which should not be ignored. However, in the case of DAGs, this constraint is not reflected in most traditional network measures suggesting that network analysis methods need to be reexamined carefully in such cases. This has been acknowledged in some contexts, the use of co-citation and bibliometric coupling, our neighbour overlap similarity measure, is well known in scientometric analysis and again it is a response to the idea that many similar papers are not always directly linked by a citation. However, enforcing the constraint has not always been taken to its logical conclusion as it requires further adaptation of existing methods.

We have developed our work in terms of a directed acyclic graph. However, the natural network representation of many data sets where there is a strong order or hierarchy is a directed

graph with a few cycles. For instance in citation data, a document can cite a document with a later publication date. In part this is because documents have several different publication dates [30]: the appearance of various electronic versions, date it was assigned an index number, formal publication date, and so forth. Such backwards citation links are regularly found, with between 0.005% and 0.4% of links reported to be in the wrong direction in various citation networks [31].

There are, however, several methods to produce an exact DAG from such data which then allows our methods to be applied. One could just delete the few links which do not respect the dominant order [31] or one could use more sophisticated approaches such as "agony" [32, 33, 34]. Our antichain method suggests another approach. Our method works in principle on directed graphs, not just those with no cycles. We exploit this in the coarse graining step of our Louvain-style numerical optimsation, see Appendix C, which can produced cycles in the derived graphs used at later stages of the method. Our approach should work well if there are few cylces. In this case, one can look at "bad links", cases where citations go from an older to a newer document or perhaps even go in both directions. The two documents connected by such a bad link will be placed in two different antichain communities. By looking at the properties of the two communities, we might be able to find the best direction for the bad link. We could even see if deleting the bad link makes most sense as indicated by a higher siblinarity score for the merger of the two antichain communities.

It is worth pointing out that although we developed our methodology in terms of DAGs, this type of partitioning can be applied to other types of networks, such as multilayer networks, bipartite networks, directed networks and even undirected networks (although in the case of the last type, antichains would most likely be composed of individual nodes).

Acknowledgement

References

[1] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[3] J. Martyn. Bibliographic coupling. *Journal of Documentation*, 20(4):236–236, 1964.

[4] H. Small. Co-citation in the scientific literature: a new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269, 1973.

[5] E.A. Leicht, G. Clarkson, K. Shedden, and M. E.J. Newman. Large-scale structure of time evolving citation networks. *The European Physical Journal B*, 59(1):75–83, 2007.

[6] M.J. Bommarito II, D. M. Katz, and J. L. Zelner. On the stability of community detection algorithms on longitudinal citation data. *Procedia - Social and Behavioral Sciences*, 4:26–37, 2010.

[7] L. Speidel, T. Takaguchi, and N. Masuda. Community detection in directed acyclic graphs. *The European Physical Journal B*, 88(8), 2015.

[8] A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, dec 1992.

[9] 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.

[10] L. Mirsky. A dual of Dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971.

[11] E.R. Gansner, E. Koutsofios, S.C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[12] A. Tarassov, N. S. Nikolov, and J. Branke. A heuristic for minimum-width graph layering with consideration of dummy nodes. In *Experimental and Efficient Algorithms*, pages 570–583. Springer Berlin Heidelberg, 2004.

[13] N. S. Nikolov and A. Tarassov. Graph layering by promotion of nodes. *Discrete Applied Mathematics*, 154(5):848–860, 2006.

[14] P. Healy and N. S. Nikolov. A branch-and-cut approach to the directed acyclic graph layering problem. In Michael T. Goodrich and Stephen G. Kobourov, editors, *Graph Drawing*, pages 98–109, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[15] Mark Hopkins. LAYERWIDTH: Analysis of a new metric for directed acyclic graphs. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002.

[16] P. Healy and N. Nikolov. Hierarchical drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 13, pages 409–454. CRC Press, 2013.

[17] V. Vasiliauskaite and T.S. Evans. Data for "Making Communities Show respect for order" paper. figshare, 2019. `10.6084/m9.figshare.9725159`.

[18] Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[19] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, 2004.

[20] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Phys. Rev. E*, 74(1):016110, 2006.

[21] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[22] D.J.de S. Price. A general theory of bibliometric and other cumulative advantage processes. *J.Amer.Soc.Inform.Sci.*, 27:292–306, 1976.

[23] Robert Ulanowicz, C Bondavalli, and M S Egnotovich. Network analysis of trophic dynamics in south florida ecosystem, fy 97: The florida bay ecosystem. *Annual Report to the United States Geological Service Biological Resources Division. Ref. No. [UMCES]CBL*, 01 1998.

[24] Austin R. Benson, David F. Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353.6295, 163–166. 2016.

[25] Jiankai Sun, Deepak Ajwani, Patrick K. Nicholson, Alessandra Sala, and Srinivasan Parthasarathy. Breaking cycles in noisy hierarchies. In *Proceedings of the 2017 ACM on Web Science Conference - WebSci '17*. ACM Press, 2017.

[26] KDD cup. `https://www.cs.cornell.edu/projects/kddcup/datasets.html`, 2003.

[27] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning*, pages 496–503, 01 2003.

[28] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, sep 2008.

[29] Lou Jost. Entropy and diversity. *Oikos*, 113(2):363–375, 2006.

[30] Stefanie Haustein, Timothy D Bowman, and Rodrigo 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.

[31] J. R. Clough, J. Gollings, T. V. Loach, and T. S. Evans. Transitive reduction of citation networks. *Journal of Complex Networks*, 3(2):189–203, 2014.

[32] Mangesh Gupte, Pravin Shankar, Jing Li, Shanmugauelayut Muthukrishnan, and Liviu Iftode. Finding hierarchy in directed online social networks. In *Proceedings of the 20th international conference on World wide web*, pages 557–566. ACM, 2011.

[33] Nikolaj Tatti. Tiers for peers: a practical algorithm for discovering hierarchy in weighted networks. *Data Mining and Knowledge Discovery*, 31(3):702–738, 2017.

[34] Elisa Letizia, Paolo Barucca, and Fabrizio Lillo. Resolution of ranking hierarchies in directed networks. *PLOS ONE*, 13(2):1–25, 02 2018.

[35] Family tree of the Greek gods. `https://en.wikipedia.org/wiki/Family_tree_of_the_Greek_gods`, Wikipedia.

[36] Python Software Foundation. Python programming language, `https://www.python.org/`.

[37] Python Packaging Authority (PyPA). Python package index (pypi) `https://pypi.org/`.

[38] Vineet Naik. pipdeptree. `https://github.com/naiquevin/pipdeptree`.

[39] Tim S. Evans, Lucille Calmon, and Vaiva Vasiliauskaite. Longest Path in the Price Model. *arXiv e-prints*, `arXiv:1903.03667`, 2019.

Appendix

A   Siblinarity antichain partition

We require a function which measures the quality of our partition of the set of nodes into our "siblinarity antichains". There are two main aspects to such a function: imposing the antichain constraint and using defining node similarity using neighbourhood overlap.

Consider a directed graph (digraph) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges, denoted (n, m) for an edge from node n to node m. Note we do not assume we have a DAG in what follows and we shall comment on this further at the end of this section.

Nodes in an antichain satisfy the condition that they are weakly connected, that is there is no directed path between the two nodes in either direction. A directed path from n to m is a sequence of nodes in which consecutive nodes are linked by an edge in the correct direction [18]. That is $\{n_j | j \in \{0, 1, \ldots, \ell\}, n_0 = n, n_\ell = m, (n_j, n_{j+1}) \in \mathcal{E}$ for $j < \ell)$. If there is a directed path in our DAG between two nodes $n, m \in \mathcal{V}$ in *either* direction i.e. the two nodes are WEAKLY CONNECTED, we will we denote this as $n \sim m$. An ANTICHAIN $\mathcal{A}$ is a subset of nodes which are not weakly connected to any of the other nodes in the same antichain, that is if $n, m \in \mathcal{A}$ then $n \nsim m$.

The second aspect is a similarity measure for two nodes, that is $\mathrm{sim}(n, m)$ is a function which increases as the nodes n and m become more similar. Our aim is to use only the information encoded in the network, information which is always available. There are still many options but in our work here we will use the number of common neighbours. That is if $\mathcal{N}(n)$ is the number of neighbours of node n then we use

$$\mathrm{sim}(n, m) = |\mathcal{N}(n) \cap \mathcal{N}(m)|. \tag{14}$$

In a DAG there are three natural sets of neighbours we can define. We can use the PREDE-CESSORS OF n, denoted $\mathcal{N}^{(\mathrm{pre})}(n)$, that is the set of nodes with outgoing edges that end at n. Alternatively, we can use the SUCCESSORS OF NODE n, denoted $\mathcal{N}^{(\mathrm{suc})}(n)$, the set of nodes connected to with incoming edges that start from n. That is

$$\mathcal{N}^{(\mathrm{pre})}(n) = \{m | (m, n) \in \mathcal{E}\}, \qquad \mathcal{N}^{(\mathrm{suc})}(n) = \{m | (n, m) \in \mathcal{E}\}, \tag{15}$$

and as illustrated in Fig. A1. Finally we can also use both sets at the same time and use $\mathcal{N}^{(\mathrm{both})}(n) = \mathcal{N}^{(\mathrm{pre})}(n) \cup \mathcal{N}^{(\mathrm{suc})}(n)$ as our neighbours set.

We then need to say if a particular value for the similarity of two nodes is large or small. To do this we define a null model, typically a randomised version of our original network, and we use the expected value in this null model for the similarity of two nodes n and m, which we will denote as $\mathrm{sim}_{\mathrm{null}}$.

Now we can put these elements together to define a function S that measures the quality of a given partition of our network into antichains, denoted as the set $\mathfrak{A}$. We consider a partition to be good if our antichains contain similar nodes. For instance in a family tree, we might want to group the biological siblings of mother and father pair. There is no direct biological connection between the siblings but they all have the same mother and father in common so the overlap in their precursor neighbour set, $\mathcal{N}^{(\mathrm{pre})}$ in the example shown in Fig. A2. We are aiming to find a partition of our set of nodes into antichains which we refer to as an ANTICHAIN PARTITION and which we denote as $\mathfrak{A}$. That is each element of the partition $\mathcal{A} \in \mathfrak{A}$ is an antichain.
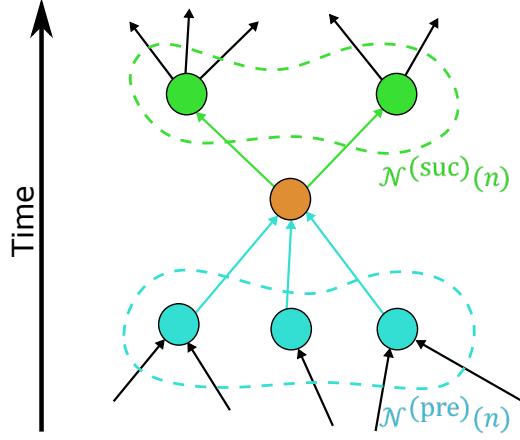
Figure A1: A figure to illustrate our neighbour set definitions.

Motivated by this family tree example, we call our quality function SIBLINARITY and we denote this $S(\mathfrak{A})$ for a given antichain partition $\mathfrak{A}$. The generic form we choose is

$$S(\mathfrak{A}) = \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n,m \in \mathcal{A}} \left(\text{sim}(n,m) - \text{sim}_{\text{null}}(n,m)\right), \quad \text{where } n \not\sim m \; \forall \; n, m \in \mathcal{A}. \tag{16}$$

Note that there is no contribution from $n = m$ in this expression. This leads to the result that $S(\mathfrak{A})$ is zero for the trivial antichain partition, the one where each nodes is in an antichain by itself, i.e. $\mathfrak{A}_{\text{trivial}}$ where

$$\mathfrak{A}_{\text{trivial}} = \{\{v\} | v \in \mathcal{V}\}. \tag{17}$$

As noted there are many possible choices for the similarity function and the null model used for comparison. In practice what we use here is given by (14) which gives us

$$S(\mathfrak{A}) = \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n,m \in \mathcal{A}} \left(|\mathcal{N}(n) \cap \mathcal{N}(m)| - \mathbb{E}(|\mathcal{N}(n) \cap \mathcal{N}(m)|)\right), \quad \text{where } n \not\sim m \; \forall \; n, m \in \mathcal{A}. \tag{18}$$

The outer sum is over all antichains $\mathcal{A}$ in the antichain partition; the inner sum is over all pairs of nodes in a given antichain $\mathcal{A}$. A contribution to the total siblinarity from a pair of nodes n and m is equal to the size of the intersection between their neighbours (predecessors or successors or perhaps both) minus the expected value of the size of this intersection, $\mathbb{E}(|\mathcal{N}(n) \cap \mathcal{N}(m)|)$. The expected value depends on the choice of the null model.

For instance, if we use a configuration model [18] as a simple null model in which the DAG has been randomised maintaining the degree of every node and the directions of the edge but otherwise the order of the DAG has been lost. Consider one term in our expression and so focus on a given pair of nodes n and m in the same antichain $\mathcal{A}$. Then pick one of the $|\mathcal{N}(n)|$ neighbours of node n, say node p. For simplicity we imagine that we are looking at successors so that this neighbour p is at the end of an edge leaving n. In our simple configuration null model this neighbouring node will have in-degree $\langle (k^{(\text{in})})^2 \rangle / \langle k^{(\text{in})} \rangle$. That is neighbouring nodes of node n have on average $(\langle (k^{(\text{in})})^2 \rangle - 1) k_n^{\text{out}} / \langle k^{(\text{in})} \rangle$ incoming edges which could be at the end of edges from node m. Given node m has k_m^{out} edges, the number of common neighbours may be estimated to be

$$\mathbb{E}(|\mathcal{N}(n) \cap \mathcal{N}(m)|) \approx \frac{(\langle (k^{(\text{in})})^2 \rangle - 1) k_n^{\text{out}} k_m^{\text{out}}}{\langle k^{(\text{in})} \rangle |\mathcal{E}|}. \tag{19}$$
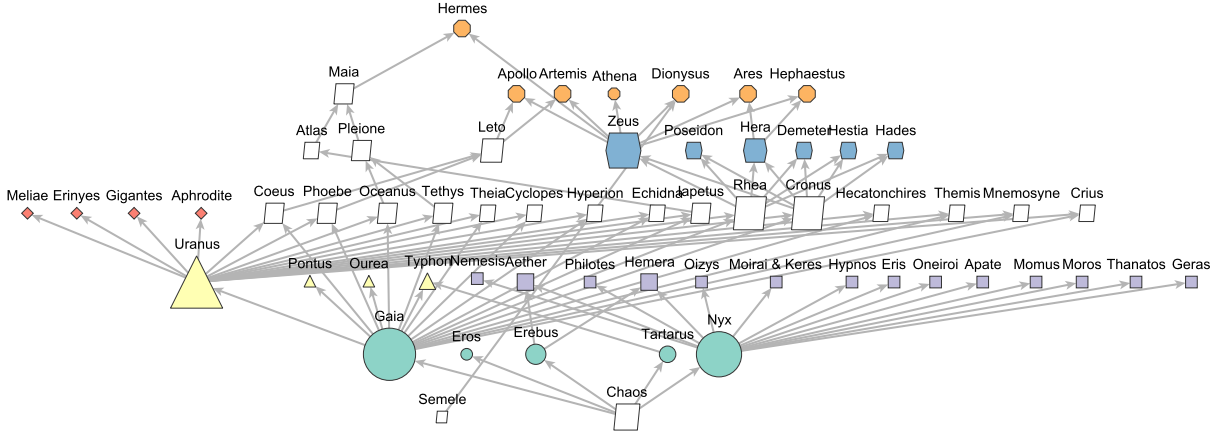
28

Figure A2: A family tree of Greek gods based on data from Wikipedia [35] (see section E.1). Links are from parents to offspring. Colours of nodes and their shapes both show grouping of deities using siblinarity based on common predecessors. For a comparison see the $\lambda = 2$ example in Fig. A3 which uses both successor and predecessor neighbours. White vertices indicate nodes in an antichain community of size one. $\lambda = 2$, size of a node indicates its total degree.

However, we do not use this null model. Rather we first rewrite our siblinarity in terms of matrices and then use that to inspire the choice of null model.

We can rewrite (18) in terms of the adjacency matrix $\mathbf{A}$ for our DAG. We use the convention that $\mathbf{A}_{nm}$ is the weight of the edge from n to m, with zero weight for no edge. Consider $\tilde{\mathbf{A}}$ as an effective similarity matrix obtained from the product of the adjacency matrix and its transpose. In the case where we have an unweighted DAG, $\tilde{\mathbf{A}}^{(\mathrm{suc})} = \mathbf{A}.\mathbf{A}^{\mathrm{T}}$ is our successors-based similarity matrix whereas $\tilde{\mathbf{A}}^{(\mathrm{pre})} = \mathbf{A}^{\mathrm{T}}.\mathbf{A}$ is a similarity matrix based on predecessors, so emulating the expressions in (15). Should we choose to use both sets of neighbours then we simply use the sum of these two matrices $\tilde{\mathbf{A}}^{(\mathrm{both})} = \tilde{\mathbf{A}}^{(\mathrm{suc})} + \tilde{\mathbf{A}}^{(\mathrm{pre})}$. Whichever of these effective similarity matrices $\tilde{\mathbf{A}}$ we use, it means we may write our siblinarity function in the following form:

$$S(\mathfrak{A}) = \sum_{n,m} \left(\tilde{A}_{nm} - \frac{\kappa_n \kappa_m}{W} \right) \delta(\mathcal{A}_n, \mathcal{A}_m), \quad \text{where } n \in \mathcal{A}_n \in \mathfrak{A},\ m \in \mathcal{A}_m \in \mathfrak{A}. \qquad (20)$$

Here the $\delta(\mathcal{A}_n, \mathcal{A}_m)$ is one if n and m are in the same antichain ($\mathcal{A}_n = \mathcal{A}_m$), zero if they are in different antichains ($\mathcal{A}_n \cap \mathcal{A}_m = 0$). We define κ_n to be the effective strength of edges attached to a node n in the similarity matrix $\tilde{\mathbf{A}}$ so $\kappa_n = \sum_m \tilde{A}_{nm}$, while and $W = \sum_{n,m} \tilde{A}_{nm}$ is the total weight of edges in the similarity matrix.

Note that the form of (20) means that we have chosen an explicit form for our null model. This matrix form (20) has been chosen to emulate the modularity function [19] used as a measure of the quality of a partition of nodes in a weighted undirected network with adjacency matrix $\tilde{\mathbf{A}}$. The null model we use is one in which we look at a "second-neighbour network" whose adjacency matrix is $\tilde{\mathbf{A}}$. This has the same nodes as the original DAG $\mathcal{G}$ with undirected but weighted edges present between nodes if the are second neighbours in the original DAG. These second neighbours in the original DAG $\mathcal{G}$ are defined using by going one step forwards and one step backwards if we are using successor neighbourhoods, and similarly for other choice of neighbourhood $\mathcal{N}$ of (15). The null model is a configuration model [18] for this

second-neighbour network.

Also note that the form given here includes non-zero diagonal entries $\tilde{A}_{nn}$ with corresponding contributions to the strength's κ_n. For instance if the original DAG $\mathcal{G}$ is unweighted then $\tilde{A}_{nn}$ is the number of (first) neighbours of node n in the DAG $\mathcal{G}$. One could eliminate the self-loops from the second-neighbour network producing a non-backtracking form for the adjaccency matrix $\tilde{\mathbf{A}}$.

The big difference between siblinarity and modularity is that for our context, our partitions are restricted to be antichains, something implicit in our $\mathfrak{A}$ notation. Apart from this important restriction, we are working on the modularity of a derived weighted but undirected network with adjacency matrix given by $\tilde{\mathbf{A}}$. If our original DAG was unweighted, this effective adjacency matrix counts the number of 'routes' (not a path in the usual precise definition used in graph theory) which consist of one forward and one backwards step on our original DAG.

A.1   Resolution

It is worth noting that we can control the resolution of obtained partition by scaling the null model contribution in the siblinarity function (16) by a parameter λ. This mimics one way that the resolution can be changed for community detection using modularity [20]. In our case we suggest a modified form for the generic siblinarity function

$$S(\mathfrak{A}, \lambda) = \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n,m \in \mathcal{A}} \left(\text{sim}(n, m) - \lambda \, \text{sim}_{\text{null}}(n, m) \right), \quad \text{where } n \not\sim m \ \forall \ n, m \in \mathcal{A}. \tag{21}$$

This clearly reduces to the original equation, (18), when $\lambda = 1$. Large values of λ would yield smaller antichains as adding nodes to an antichain produces a penalty. So for large enough λ, the antichain partition which maximises the modified siblinarity will be where each antichain contains just one node. When λ is zero, any two nodes which are not connected by a path but share at least one shared neighbour will increase the siblinarity value if put in the same antichain. So for small λ we expect the maximal modified siblinarity is likely to be something that has the fewest number of antichains. In particular, a negative λ will allow nodes with no path between them and with no common neighbours to to have larger siblinarity values if they are in the same antichain rather than each node being in an antichain of one node.

To illustrate this idea, we find antichain communities by maximising a modified form of our usual matrix siblinarity (20), namely

$$S(\mathfrak{A}, \lambda) = \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n \in \mathcal{A}} \sum_{m \in \mathcal{A} \backslash n} \left(\tilde{A}_{nm} - \lambda \frac{\kappa_n \kappa_m}{W} \right). \tag{22}$$

We use $S(\mathfrak{A}, \lambda)$ to find antichain communities in the family tree of the Greek gods [35] (see section E.1) shown in Fig. A3. As expected, when λ is small, such as in A, communities are large and very close to height antichains. Larger values of λ in B and C produce more refined partitions. The same behaviour can be sen in a very simple example in section B.
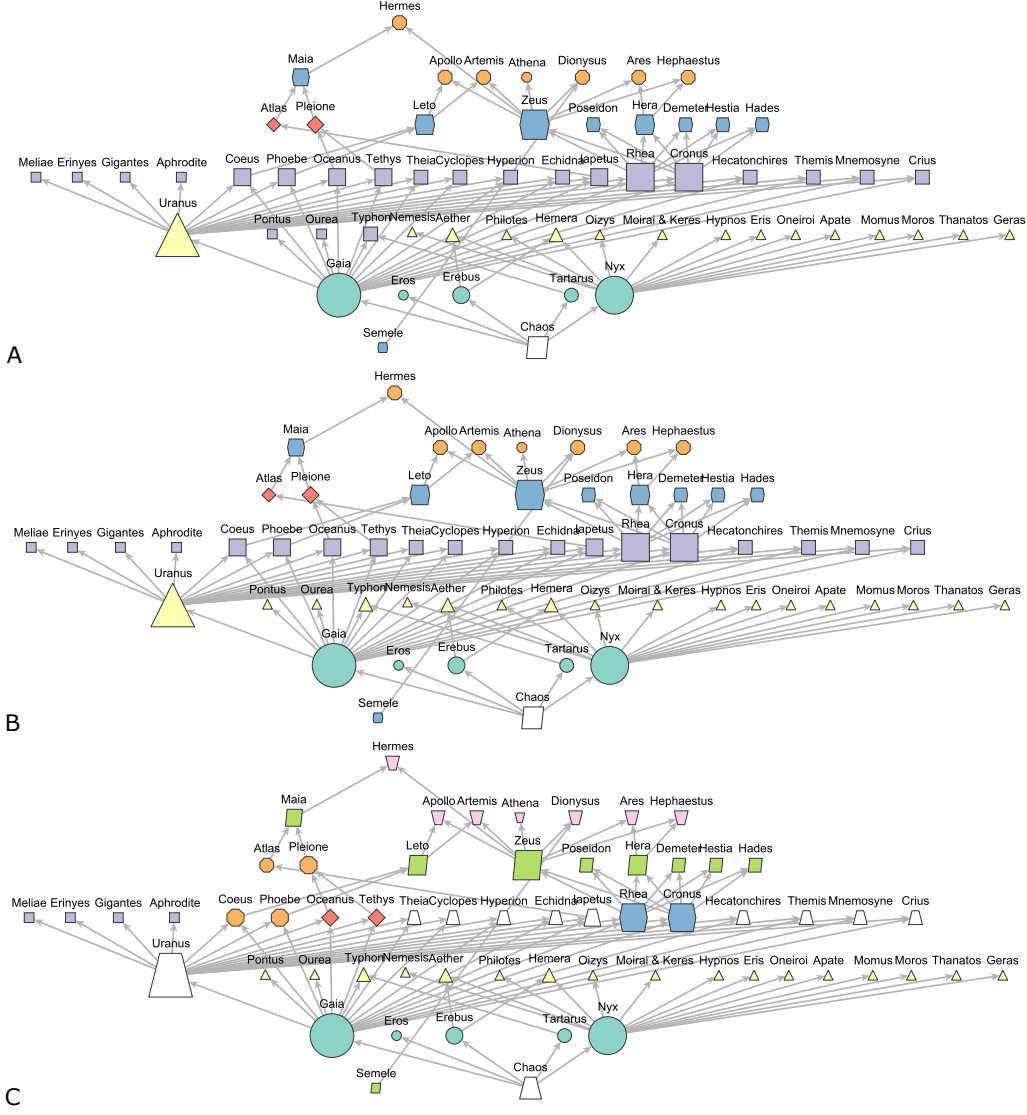
Figure A3: Antichains in the family tree of Greek Gods [35] (see section E.1), obtained by maximising the modified siblinarity $S(\mathfrak{A}, \lambda)$ (22) using different resolution parameters λ: in A $\lambda = 0.5$, in B $\lambda = 1.5$, in C $\lambda = 2$. Different colours an shapes show different siblinarity communities, obtained by considering both, future and past neighbours of nodes. As expected, when λ is small, communities are large and very close to height antichains. Larger values of λ produce more refined partitions. White vertices indicate nodes in an antichain community of size one. Note that we can compare this $\lambda = 2$ example against the antichain communities in Fig. A2 as the latter uses predecessors unlike here where both successor and predecessor neighbours are used to evaluate siblinarity.

B   A Simple example

Consider the network G shown in the centre of Fig. B4. Numbering the vertices from the

$\mathcal{H}$ for $\mathfrak{A}_2$
$\{\{1\}, \{2,3\}, \{4,5\}, \{6\}\}$.

$\mathcal{G}$

$\mathcal{H}$ for $\mathfrak{A}_3$
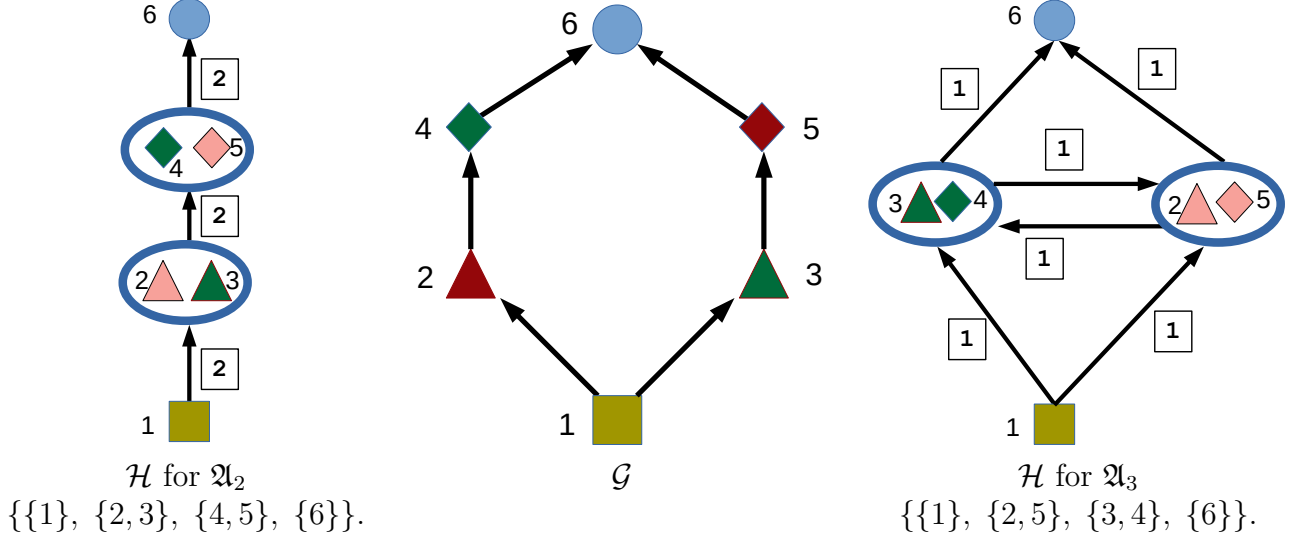$\{\{1\}, \{2,5\}, \{3,4\}, \{6\}\}$.

Figure B4: In the centre the simple example G considered in the text. On the left is the reduced network $\tilde{G}_2$ based on the height antichain partition $\mathfrak{A}_2$ of (28). On the right is the reduced network $\tilde{G}_3$ based on the antichain partition $\mathfrak{A}_3$ of (30). Note that this last antichain partition $\mathfrak{A}_3$ produces a reduced network which is a directed graph with cycles. In the networks on the left and right, the numbers in squares indicate the weight of the nearest edge. The central network is unweighted. For the reduced networks on the left and right, the ovals are the nodes in the centre but within they show which vertices were merged to form the new node in the reduced graph.

bottom up we have edges $(1,2)$, $(1,3)$, $(2,4)$, $(3,5)$, $(4,6)$, and $(5,6)$ where (m,n) is an edge from vertex m to vertex n. Using a convention that A_{mn} is an edge from m to n we have that columns are labelled by n and rows are labelled by m. So G in Fig. B4 has adjacency matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{23}$$

In this example we will use the following definition of siblinarity

$$\hat{S}(\mathfrak{A}) = \sum_{\mathcal{A} \in \mathfrak{A}} \sum_{n \in \mathcal{A}} \sum_{m \in \mathcal{A}} \left(\tilde{A}_{nm} - \frac{\kappa_n \kappa_m}{W} \right), \tag{24}$$

$$\kappa_n := \sum_m \tilde{A}_{nm}, \quad W = \sum_{n,m} \tilde{A}_{nm}.$$

Note that we are not excluding $m = n$ in this definition unlike the definition of $S(\mathfrak{A})$ of (5) but the difference is a constant independent of $\mathfrak{A}$. Since $S(\mathfrak{A})$ of (5) is zero for the trivial antichain

partition $\mathfrak{A}_{\text{trivial}}$ of (17), in which every node is placed in their own element of the partition, this is difference is equal to the value of $\hat{S}(\mathfrak{A})$ for that trivial antichain partition. In our case this is $\hat{S}(\mathfrak{A}_1)$ below. We will choose to use a successor form for the two-step matrix in (24), that is $\tilde{\mathbf{A}}^{(\text{suc})} = \mathbf{A}.\mathbf{A}^{\mathrm{T}}$, so that

$$\tilde{\mathbf{A}}^{(\text{suc})} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{25}$$

Each vertex forms on antichain

Now suppose we use this for a partition where each element of the partition is a single vertex. This is always an antichain partition. Here we have

$$\mathfrak{A}_1 = \{\{1\},\ \{2\},\ \{3\},\ \{4\},\ \{5\},\ \{6\}\} \tag{26}$$

and we find that the siblinarity value is

$$\begin{aligned}
\hat{S}(\mathfrak{A}_1) &= \left(2 - \lambda\frac{2.2}{8}\right) + \left(1 - \lambda\frac{1.1}{8}\right) + \left(1 - \lambda\frac{1.1}{8}\right) + \left(1 - \lambda\frac{2.2}{8}\right) + \left(1 - \lambda\frac{2.2}{8}\right) + 0 \\
&= 6 - \lambda\frac{14}{8} \ . \tag{27}
\end{aligned}$$

The Height Antichain Partition

The height antichain partition (same as the depth antichain partition here) is

$$\mathfrak{A}_2 = \{\{1\},\ \{2,3\},\ \{4,5\},\ \{6\}\} \ . \tag{28}$$

This has siblinarity equal to

$$\hat{S}(\mathfrak{A}_2) = \left(2 - \lambda\frac{2.2}{8}\right) + \left(2 - \lambda 3.\frac{1.1}{8}\right) + \left(4 - \lambda 3.\frac{2.2}{8}\right) + 0 = 6 - \lambda\frac{19}{8} \ . \tag{29}$$

Antichain Partition with Cyclic Derived Graph

For an antichain partition of

$$\mathfrak{A}_3 = \{\{1\},\ \{2,5\},\ \{3,4\},\ \{6\}\} \tag{30}$$

the derived graph $\tilde{G}$ will be directed but not cyclic as shown in Fig. B4. The siblinarity for this antichain partition is

$$\begin{aligned}
\hat{S}(\mathfrak{A}_3) &= \left(2 - \lambda\frac{2.2}{8}\right) + \left(2 - \lambda[\frac{1.1}{8} + \frac{1.2}{8} + \frac{2.2}{8}]\right) + \left(2 - \lambda[\frac{1.1}{8} + \frac{1.2}{8} + \frac{2.2}{8}]\right) + 0 \\
&= 6 - \lambda\frac{18}{8} \ . \tag{31}
\end{aligned}$$

Antichain Partition Four

Another possible antichain partition is

$$\mathfrak{A}_4 = \{\{1\}, \{2\}, \{3\}, \{4,5\}, \{6\}\} \tag{32}$$

This gives

$$\hat{S}(\mathfrak{A}_4) = \left(2 - \lambda\frac{2.2}{8}\right) + \left(1 - \lambda\frac{1.1}{8}\right) + \left(1 - \lambda\frac{1.1}{8}\right) + \left(3 - \lambda 3.\frac{2.2}{8}\right) + 0 = 7 - \lambda\frac{18}{8}. \tag{33}$$

Antichain Partition Five

The fifth antichain partition we consider is

$$\mathfrak{A}_5 = \{\{1\}, \{2,3\}, \{4\}, \{5\}, \{6\}\} \tag{34}$$

This is similar to antichain four but it is not related by any symmetry so gives a different result.

$$\hat{S}(\mathfrak{A}_5) = \left(2 - \lambda\frac{2.2}{8}\right) + \left(2 - \lambda 3.\frac{1.1}{8}\right) + \left(1 - \lambda\frac{2.2}{8}\right) + \left(1 - \lambda\frac{2.2}{8}\right) + 0 = 6 - \lambda\frac{15}{8}. \tag{35}$$

Antichain Partition Six and Seven

The last two possible antichain partitions have the same siblinarity by symmetry

$$\mathfrak{A}_6 = \{\{1\}, \{2,5\}, \{3\}, \{4\}, \{6\}\} \tag{36}$$

and

$$\mathfrak{A}_7 = \{\{1\}, \{2\}, \{3,4\}, \{5\}, \{6\}\} \tag{37}$$

These have siblinarity values of

$$\begin{aligned}
\hat{S}(\mathfrak{A}_6) = \hat{S}(\mathfrak{A}_7) &= \left(2 - \lambda\frac{2.2}{8}\right) + \left(1 - \lambda\frac{1.1}{8}\right) + \left(2 - \lambda[\frac{1.1}{8} + \frac{1.2}{8} + \frac{2.2}{8}]\right) \\
&\quad + \left(1 - \lambda\frac{2.2}{8}\right) + 0 \\
&= 6 - \lambda\frac{16}{8}. \tag{38}
\end{aligned}$$

Siblinarity Maximisation

The results for all the possible antichains for the DAG G of Fig. B4 are summarised in Table B1. The best partition for maximum siblinarity is not in fact the height partition $\mathfrak{A}_2$, (this is second best) but is the fourth partition $\mathfrak{A}_4$. This because the two antichains $\{2\}$ and $\{3\}$ are preferred to the single antichain $\{2,3\}$ and this is because nodes 2 and 3 have no common successors. Had we used a siblinarity that involved predecessors in some way then $\{2,3\}$ would have been preferred.

When we introduce the resolution parameter λ in (22), the modified siblinarity gives the height antichain community $\mathfrak{A}_2$ (28), a maximal antichain partition, as the best solution for $\lambda < 0$. For $0 < \lambda < 2$ we find the fourth antichain $\mathfrak{A}_4$ of (32) is the optimal. The trivial partition with each antichain containing one node, $\mathfrak{A}_1$ of (26), has the largest modified siblinarity for $\lambda > 2$.

Antichain Partition		Siblinarity		Mod.Siblinarity	
		$\hat{S}(\mathfrak{A})$	$S(\mathfrak{A})$	$\hat{S}(\mathfrak{A},\lambda)$	$S(\mathfrak{A},\lambda)$
$\mathfrak{A}_1$	$\{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\}\}$	$\frac{34}{8}$	$\frac{0}{8}$	$6-\lambda\frac{14}{8}$	0
$\mathfrak{A}_2$	$\{\{1\},\{2,3\},\{4,5\},\{6\}\}$	$\frac{37}{8}$	$+\frac{3}{8}$	$7-\lambda\frac{19}{8}$	$1-\lambda\frac{5}{8}$
$\mathfrak{A}_3$	$\{\{1\},\{2,5\},\{3,4\},\{6\}\}$	$\frac{30}{8}$	$-\frac{4}{8}$	$6-\lambda\frac{18}{8}$	$-\lambda\frac{4}{8}$
$\mathfrak{A}_4$	$\{\{1\},\{2\},\{3\},\{4,5\},\{6\}\}$	$\frac{38}{8}$	$+\frac{4}{8}$	$7-\lambda\frac{18}{8}$	$1-\lambda\frac{4}{8}$
$\mathfrak{A}_5$	$\{\{1\},\{2,3\},\{4\},\{5\},\{6\}\}$	$\frac{33}{8}$	$-\frac{1}{8}$	$6-\lambda\frac{15}{8}$	$-\lambda\frac{1}{8}$
$\mathfrak{A}_6$	$\{\{1\},\{2,5\},\{3\},\{4\},\{6\}\}$	$\frac{32}{8}$	$-\frac{2}{8}$	$6-\lambda\frac{16}{8}$	$-\lambda\frac{2}{8}$
$\mathfrak{A}_7$	$\{\{1\},\{2\},\{3,4\},\{5\},\{6\}\}$	$\frac{32}{8}$	$-\frac{4}{8}$	$6-\lambda\frac{16}{8}$	$-\lambda\frac{2}{8}$

Table B1: A table of siblinarity values using $\hat{S}(\mathfrak{A})$ of (24) and $S(\mathfrak{A},\lambda)$ of (22) for different antichain partitions of the graph G of Fig. B4. Note that the values of siblinarity $S(\mathfrak{A})$ defined in (5) are simply $S(\mathfrak{A}) = \hat{S}(\mathfrak{A}) - \hat{S}(\mathfrak{A}_1)$ and similarly for the modified siblinarity values.

C   Louvain Siblinarity Optimisation

Having defined a quality function, we can look for a partition of our nodes into antichains, $\mathfrak{A}$, which maximises the siblinarity $S(\mathfrak{A})$. This task faces the same challenges as most network community detection methods; there are many local minima and only approximate solutions can be found in a reasonable amount of computational time. Here, we will discuss how to adapt the Louvain algorithm [21] which is a widely used and successful methods to find communities in networks which maximise modularity. Emulating the Louvain algorithm, our siblinarity maximisation method is an iterative greedy algorithm in which each iteration has two phases.

In the first phase of our algorithm, we start with an initial partition into antichains in which each node is assigned to its own antichain. At each subsequent step, we try to move a single node n from its current antichain, $\mathcal{A}_a$, to another antichain $\mathcal{A}_b$, always choosing the configuration which maximises the siblinarity, even if that means leaving the antichains unchanged. In our implementation, we visit each node in a fixed sequence. Once the sequence is exhausted, we sweep through the same sequence once again. This process is continued until there are no more changes in the optimal antichain partition possible whatever node n we choose to examine. That is when changing the antichain partition by moving just one node can not increase the siblinarity. This marks the end of the first phase. In principle, we can also stop this first phase at any point as every new antichain partition is, by definition better than the last. So in our algorithm we also stop this phase if we have completed a given number of sweeps since the second phase is almost certain to simplify the problem and so speed up subsequent iterations.

For each node n we choose for a possible move, the change in siblinarity is calculated for removing n from its current antichain, $\mathcal{A}_a$, and placing in a new antichain A_b. It is important to enforce the constraint that the node n must not be connected to any existing node m in the potential new antichain $\mathcal{A}_b$, i.e. we want $\mathcal{A}_b \cup \{n\}$ to be an antichain. In our algorithm, we further limit the choice of which new antichains $\mathcal{A}_b$ we examine. If we are using siblinarity defined using a similarity measure using a neighbourhood set $\mathcal{N}(n)$ for our nodes n, then we look for antichains $\mathcal{A}_b$ which contain at least one node $m \in \mathcal{A}_b$ which has a non-trivial similarity

measure with our chosen node n, i.e. for us we require $|\mathcal{N}(n) \cap \mathcal{N}(m)| > 0$. These potential new antichains for node n are easy to find as this involves a two-step walk on the network starting from n. If we use a neighbourhood based on successors, that is $\mathcal{N}^{(\mathrm{suc})}n$, then we only look at antichains $\mathcal{A}_b$ which contain a predecessor m of a successor of n. We will call these SUCCESSOR ANTICHAINS. If we look only at predecessors neighbourhoods and so use $\mathcal{N}^{(\mathrm{pre})}(n)$ for our neighbourhood sets, we shall refer to the resulting antichain partitions as PREDECESSOR ANTICHAINS. There is only one other case we examine, and that is we also check the case where we allow n to join a new antichain consisting of the node n alone, i.e. $\mathcal{A}_b = \emptyset$.

The change in siblinarity ΔS is given by

$$\Delta S(\mathcal{A}_a, \mathcal{A}_b \to \mathcal{A}_a \backslash n, \mathcal{A}_b \cup \{n\}) = \sum_{m \in \mathcal{A}_b} \left(|\mathcal{N}(n) \cap \mathcal{N}(m)| - \mathbb{E}(|\mathcal{N}(n) \cap \mathcal{N}(m)|) \right)$$
$$- \sum_{q \in \mathcal{A}_a \backslash n} \left(|\mathcal{N}(n) \cap \mathcal{N}(q)| - \mathbb{E}(|\mathcal{N}(n) \cap \mathcal{N}(q)|) \right), \qquad (39)$$
$$\text{provided } n \not\sim \mathcal{A}_b.$$

The first term is the contribution from the addition of node n to the antichain $\mathcal{A}_b$, while the second term is the effect of the removal of node n from its current antichain $\mathcal{A}_a$. Note the condition that n is not connected to any node in the existing antichain $\mathcal{A}_b$ which we denote as $n \not\sim \mathcal{A}_b$. This is needed to ensure $\mathcal{A}_b$ remains an antichain when n is added.

In the matrix notation of (20), the change in siblinarity is given by

$$\Delta S(\mathcal{A}_a, \mathcal{A}_b \to \mathcal{A}_a \backslash n, \mathcal{A}_b \cup \{n\}) = \sum_{m \in \mathcal{A}_b} \left(\tilde{A}_{nm} - \frac{\kappa_n \kappa_m}{W} \right) - \sum_{q \in \mathcal{A}_a \backslash n} \left(\tilde{A}_{nq} - \frac{\kappa_n \kappa_q}{W} \right)$$
$$\text{provided } n \not\sim \mathcal{A}_b. \qquad (40)$$

In the second phase we create an *induced graph* $\mathcal{H} = \{\mathcal{V}_{\mathrm{H}}, \mathcal{E}_{\mathrm{H}}\}$ from the original graph $\mathcal{G}$ and the antichain partition $\mathfrak{A}$ left at the end of phase one. Each node $a \in \mathcal{V}_{\mathrm{H}}$ in this induced graph $\mathcal{H}$ represents a single antichain, $\mathcal{A}_a \in \mathfrak{A}$, as given at the end of the previous phase. The edges between nodes of induced graph are given a weight equal to the sum of the weights of all the edges between the equivalent antichain nodes in the original graph $\mathcal{G}$ of the induced graph. For instance, if there were k_{ba} edges all of weight 1 pointing from nodes in the antichain $\mathcal{A}_a$ to the antichain $\mathcal{A}_b$ at the end of phase one, there would be a directed edge $(a, b) \in \mathcal{E}_{\mathrm{H}}$ in the induced graph with weight equal to k_{ba} in the induced graph[5]. In terms of matrices, if H_{ba} is equal to the weight of the edge from node a to b in the adjacency matrix for the induced graph, then we have that

$$H_{ba} = \sum_{m \in \mathcal{A}_a} \sum_{n \in \mathcal{A}_b} A_{nm}. \qquad (41)$$

Once the induced graph is created, the algorithm continues by applying finding an antichain partition of the induced graph using siblinarity, starting with the phase one. The algorithm continues until there is no substantial increase in the siblinarity function (18).

---

[5]Induced graph does not have to be a DAG: antichains are possible in graphs with cycles as Fig. B4 shows. By definition, an antichain is a subset of nodes such that there is no path between any of two of them in this subset. This is perfectly valid in any graph, however, in some they are more interesting than in others.

D Basic Statistics on Antichains

One way to look at antichains is to look at bipartite network $\mathcal{B}(\mathcal{A})$ associated with each antichain. For each antichain $\mathcal{A}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define a bipartite network $\mathcal{B}(\mathcal{A})$. The first type of vertex in the bipartite network are simply those in the antichain itself, $\mathcal{A}$. The second type of vertex in $\mathcal{B}(\mathcal{A})$ is the set of all neighbours of the antichain vertices, that is

$$\mathcal{N}(\mathcal{A}) = \cup_{v \in \mathcal{A}} \mathcal{N}^{(\mathrm{suc})}(v) \,. \tag{42}$$

An edge of the bipartite graph $\mathcal{B}(\mathcal{A})$ is present if there is a corresponding connection between a given vertex in the antichain $\mathcal{A}$ to any of its neighbours, any vertex in $\mathcal{N}(\mathcal{A})$

$$\mathcal{E}(\mathcal{A}) = \{(v,n) | n \in \mathcal{N}(\mathcal{A}), v \in \mathcal{A}, (v,n) \in \mathcal{E}\} \cup \{(n,v) | n \in \mathcal{N}(\mathcal{A}), v \in \mathcal{A}, (v,n) \in \mathcal{E}\} \,. \tag{43}$$

In practice, we also remove loosely connected neighbour nodes but for simplicity we will not indicate that in our definitions here. This gives us $\mathcal{B}(\mathcal{A}) = (\mathcal{A} \cup \mathcal{N}(\mathcal{A}), \mathcal{E}(\mathcal{A}))$.

There are many possible network measurements we can make on each bipartite graph which can help us understand the nature of the antichains found in any example. In our work we focussed on some of the simplest measures.

- $|\mathcal{A}|$ — The size of an antichain, i.e. the number of nodes in an antichain.

- $|\mathcal{N}|$ — The number of neighbours of an antichain.

- $\langle k \rangle$ — The average number of neighbours of nodes in an antichain.

$$\langle k \rangle (\mathcal{A}) = \frac{|\mathcal{E}(\mathcal{A})|}{|\mathcal{A}|} \tag{44}$$

- $\sigma(k)$ — The standard deviation of the number of neighbours of nodes in an antichain.

$$\sigma k(\mathcal{A}) = \frac{|\mathcal{E}(\mathcal{A})|}{|\mathcal{N}|} \tag{45}$$

- $\langle k \rangle / |\mathcal{N}| = \frac{|\mathcal{E}(\mathcal{A})|}{|\mathcal{A}||\mathcal{N}|}$ — This is the density of the bipartite graph, the number of edges dived by the maximum number possible.

The average degree and standard deviation of degree of nodes in the antichain, $\langle k \rangle$ and $\sigma(k)$, give us a picture of how the even the connections between antichain nodes and their neighbours are. If $\langle k \rangle \approx |\mathcal{N}|$ and $\sigma(k) \approx 0$, we know that nodes of very similar degrees are joined together in an antichain, and their neighbourhoods are largely overlapping.

The ratio between $\langle k \rangle$ and $|\mathcal{N}|$ tell us how similar our bipartite graph is to a sparse "zig-zag" pattern. If $\langle k \rangle$ is small in comparison to $\mathcal{N}$, then we can expect small overlap overall between all nodes in the antichain. This statistic approaches 1 if every node in the antichain is connected to every neighbour.

E Additional Examples of Siblinarity and Data

E.1 Greek Gods

We have named the function used to find antichain partitions "siblinarity" using the analogy with a family tree. We use the number of common neighbours to link nodes in an antichain just as siblings share biological parents. Since family trees based on biological parentage are predominantly trees, directed acyclic graphs but which have few if any loops in the undirected version, we have looked to fiction to provide a more interesting example of a family tree. We have taken information on Greek gods from Wikipedia [35] with each node representing a Greek god with edges from a god to a child. Fig. A2 shows the results of applying our siblinarity clustering siblinarity based on common predecessors to this data set. See [17] for the original data.

E.2 Python Dependencies

Many software programmes can be extended by adding packages, extra programmes which extend the functionality of the core package. As the number of packages grows, some of the added packages start to use the functionality of some of the other packages added to the original core application. In order to ensure that each package can run correctly, there is usually a system to noting the dependencies of each package, that is, what other packages are required in order to run any one package.

A good example of such an software 'ecosystem' is the computer language `python` [36]. The `PyPI` repository [37] of packages for python records over 180,000 projects at the time of writing. To illustrate the principle, we used a python `python` installation on one of the author's machines in 2019. We used V.Naik's `pipdeptree` [38] (itself a python package listed on `PyPI`) to extract the directed acyclic graph representing the package dependency DAG. Each node is a package and we add a link from the parent package to the sibling, where the sibling package is requires the installation of the parent in order to work. The results are shown in Fig. E5 and the data used is provide on [17].

This illustrates several of the points made elsewhere. For instance, the `scipy`, `pandas`, and `matplotlib` packages on the far right of Fig. E5, are in the same antichain cluster. These are often used for scientific analysis but they provide different functionality: scientific functions, data handling and plotting respectively. They share many predecessors, such as `numpy`, since they all need to handle large quantities of numerical values efficiently. It is interesting to see that these three packages are not all at the same height so a height based clustering would not put them together. On the other hand `sphinx` (a tool for producing documentation seen on the far left of Fig. E5) is at the same height as `pandas`, and `matplotlib` but is placed in a different cluster as the packages share so few common predecessors.
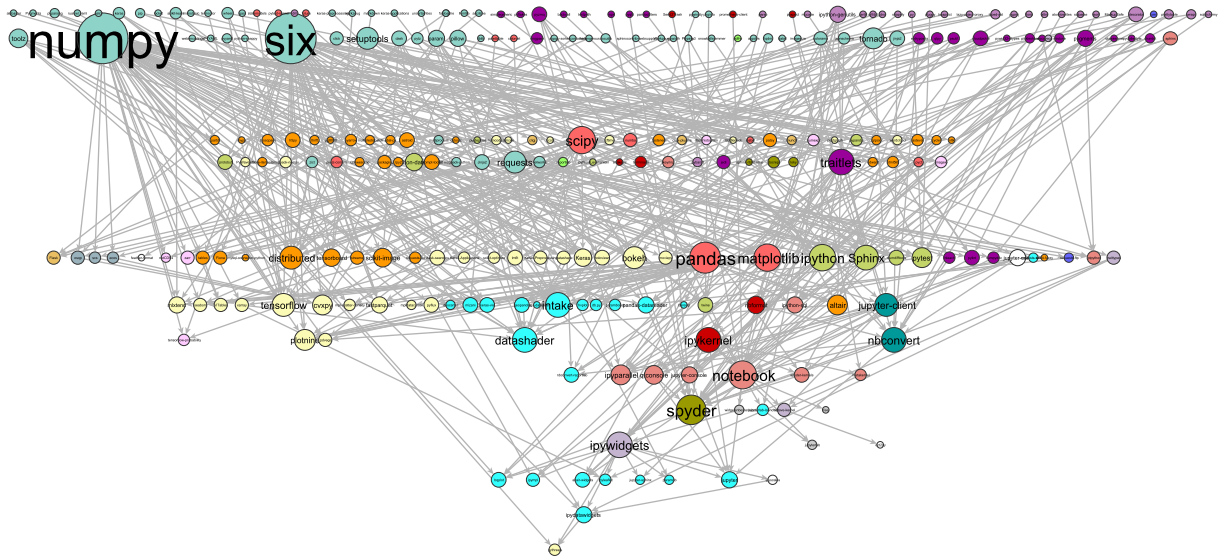
Figure E5: The dependencies of `python` packages on one of the author's machines in 2019. Each node is a package. Links are to the dependent package and are from the parent package needed for the siblings to function, no transitive reduction has been performed. Colours indicate the different antichain communities found using siblinarity based on both successors and predecessors. White indicates a node in a cluster by itself. The size of the node and the size of the labels is related to the degree of each node. The vertical positioning is given by the depth of a node with small variations in the top two depths to improve visibility. An electronic version of this file is available on [17] which will allow readers to see the names of packages with small labels.

F Price Model with Subject Fields

The Price model for citation networks [22] produces a DAG with a fat-tailed (power-law) distribution for the out-degree of nodes in our conventions which represents the citation count of papers. We modify the Price model by assigning each paper to a 'field' and the edges, citations between papers, are biassed so they are usually between papers in the same field. While an unrealistic model of citation networks in many ways, it contains three key features of real citation networks: the order of papers imposed by time, the fat-tailed citation count distribution, and the preference of most papers to cite papers within a similar field. We use it as a DAG with a planted partition to enable us to make controlled comparisons between the different community detection approaches discussed.

The model defines a sequence of networks $\mathcal{G}(t)$ where t is a positive integer playing the role of time and which gives us an order to the nodes in our networks. Each graph $\mathcal{G}(t)$ has t nodes with vertex set $\mathcal{V}(t)$ and edge set $\mathcal{E}(t)$. In our notation, the node $u(s)$ is always the node added at step s in the process, so $u(s) \in \mathcal{V}(t)$ provided $0 < s \leq t$.

The nodes in these networks are also partitioned into different fields, that is each node $u(t)$ is in one of F fields. The fields will be labelled by integers between 0 and $(F-1)$ with $f(t) \in \{0, 1 \ldots, (F-1)\}$ denoting the field of node $u(t)$. This creates a sequence of partitions $\mathfrak{F}(t)$ of our nodes where $\mathfrak{F}(t) = \{\mathcal{F}_f(t)|f \in \{0, 1, \ldots, (F-1)\}\}$ and $\mathcal{F}_f(t) \subseteq \mathcal{V}(t)$. A node $u(s)$, for $0 < s \leq t$, belongs to a element $\mathcal{F}_{f(s)}(t) \in \mathfrak{F}(t)$, the set of papers at time t in the same field $f(s)$ as the paper published at time s.

To create the next graph in the sequence, $\mathcal{G}(t+1)$, we first add a new node $u(t+1)$ to the vertex set, so $\mathcal{V}(t+1) = \mathcal{V}(t) \cup \{u_{t+1}\}$. This new node is assigned to the next field, $f = |(t+1)|_F$, so that $u(t+1)$ is in $\mathcal{F}_f(t+1) = \mathcal{F}_f(t) \cup \{u(t+1)\}$ but the other elements in the field partition remain unchanged, $\mathcal{F}_g(t=1) = \mathcal{F}_g(t) \forall g \neq f$.

We now add m directed edges to this new node $v(t+1)$ from existing nodes $u(s)$ where $s < t$. To encode the "cumulative advantage" principle of Price, that is the higher the current citation count of a paper the more likely it is to be cited, we can chose nodes $u(s)$ from the existing nodes $\mathcal{V}(t)$ in the network $\mathcal{G}(t)$ with probability $\Pi^{(\mathrm{CA})}(t, s)$ defined as[6]

$$\Pi^{(\mathrm{CA})}(t, s) = \frac{k^{\mathrm{out}}(t, s) + 1}{|\mathcal{V}(t)| + |\mathcal{E}(t)|}. \tag{46}$$

Here $k^{\mathrm{out}}(t, s)$ is the number of outgoing edges from node $u(s)$ in $\mathcal{G}(t)$, the network at time t. In our conventions, these edges represent citations from later papers to the paper published at time s. The planted partition representing the fields is used on top of the cumulative advantage in $\Pi^{(\mathrm{CA})}(t, s)$ by ensuring that a fraction ϕ of the edges are chosen to lie between nodes in the same field. So the overall probability for choosing an existing node $u(s)$ as the source of an edge to new node $v(t)$ is, to a good approximation[7],

$$\Pi(t, s) \approx \left(\phi F \delta(f(s), f(t+1)) + (1-\phi)\frac{F}{(F-1)}(1 - \delta(f(s), f(t+1))) \right) \Pi^{(\mathrm{CA})}(t, s). \tag{47}$$

Note that we also impose the constraint that there is at most only one edge between any two nodes and we choose the initial graph to be a transitively complete graph of size $m + 1$.

---

[6]Other forms linear in $k^{\mathrm{out}}(t, s)$ are also easy to work with (for example see [18]), but these variations are not our focus here. We chose to follow the same form as used in Price's original paper.

[7]For instance, we have assumed that the fraction of nodes in any one field is always $1/F$ and that the degree distribution is the same for all fields at all times. These are good approximations at later times.

Neither of these constraints will have any significant effect on the measurements we make for the large networks we use in our work here.