COLORED DIRECTED ACYCLIC GRAPHS FOR MULTI-CLASS ATTRIBUTION NETWORKS

M. STEPHENSON, M. ZARGHAM

Abstract. This research note discusses the creation of a Directed Acyclic Memex, a graph theoretic tool helping to enable one of the foundational concepts of the internet. The practice of research citing earlier work is well represented by a directed graph structure price1965networks, and we suggest that a natural addition to this process is the ability to "color" paths, groups, edges, or nodes. This allows for individuals to create "associative trails" of upstream occurrences, and to easily classify them along relevant dimensions. Citations of ideas are a natural use case, but the concept is quite general and can be used for accounting, collaboration, and distribution.

1. Introduction

The person who oversaw landmark scientific projects like the Manhattan Project, the National Science Foundation (NSF), and NASA is the same person widely credited with inspiring the internet. This is not a coincidence. This figure, Vannevar Bush, sought to fund free and open research because he believed that science and progress resulted from "the free play of free intellects, working on subjects of their own choice, in the manner dictated by their curiosity for exploration of the unknown". But so too did he think that computers would offer an incredible tool for helping aid this exploration and play– he wrote a 1945 article called "As We May Think" that internet pioneers like Ted Nelson and Douglas Engelbart have cited as a formative influence on internet.

Writing well before the computer age had taken hold, Bush believed that even then published knowledge was already "far beyond" our ability to make use of it. He proposed the Memex, a tool which would enable a more efficient organization of human thought. Users could navigate and share information by sharing "associative trails"—paths through articles which inspired and validated certain ideas. An important project inspired the Memex, Ted Nelson's Xanadu project, was ambitious in scope but was never fully realized. The vision of the internet found in the Memex and the Xanadu Project have thus far foundered on incentive and design problems[1], but recent innovations may make such a vision newly possible. ++ references from abstract

Date: Aug 29, 2018.

[1]As investor Peter Thiel notes, "Their technology probably would have worked at scale, but it could have worked only at scale". thiel2014zero Moreover, Nelson struggled "to balance openness with compensation and recognition for himself and the other early pioneers", quoraxanadu which are the sorts of problems endemic to all open innovation. nelson1959simple, stephensonplanck

This paper proposes a formalism: "Colored Directed Acyclic Graphs", and motivates them specifically for a broad class of problems we call "Multi-class attribution networks". We suggest that these classes, along with other complementary incentive mechanisms, can help to realize a very natural structure of information across networks—a move toward a "Directed Acyclic Memex" (DAM), one of the original visions of computing technologies as enhancements of thought.

Colored Directed Acyclic Graph Part of intro (theory) Nodes and edges are basic abstractions of graph theory. Edges exist as connections among nodes, in a broadly analogous way to how friendships exist as connections among people. You can mentally build the components of such a graph by thinking of individuals as nodes and dispositions of friendship as edges—people are dots, and friendships are lines connecting them. Friendship is usually reciprocated so the edges connecting two reciprocal friends will have arrows at each end. But if someone thinks of you as a friend and you, perhaps, think of them as something else (an acquaintance, perhaps) the edge between you point only to you, a one-way edge. Nodes connected exclusively by one-way edges create a directed graph, and the fact that each individual is unique and has unique edges, means that the graph is "acyclic" and doesn't repeat as you traverse it. This is a Directed Acyclic Graph (DAG).

A DAG of friendships would be a truly dystopian vision— a topography of vast unrequited social longing. But DAGs are a beautiful fit for attribution, because it's natural to only attribute things which already exist. Whatever underpins the second law of thermodynamics and the arrow of time, it's fairly natural to attribute as causes only those things which temporally preceded the caused. Thus the edges of our attribution DAG only point upstream, to earlier ideas or events. This appropriateness of a DAG structure for graphing scientific citations was recognized by an early historian of science price1965networks.

short paragraph outlineing the technical content

2. Notation and Definitions

In this section, the formal notation for graph theoretic, cryptographic, and data modeling concepts are presented as the foundation on which the practical construct is built. Directed Acyclic Graphs (DAGs), graph colorings, meta-data schemas, Hash Functions, content-addressibility, partial orderings are covered, directed root trees ans associated attribution metrics are covered.

2.1. Colored Directed Acyclic Graph.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote an directed graph with n vertices, m edges, and no self-loops. We denote by $\mathcal{V}(\mathcal{G}) = \{v_1, \ldots, v_n\}$ the set of vertices and by $\mathcal{E}(\mathcal{G}) \subseteq \mathcal{V}(\mathcal{G}) \times \mathcal{V}(\mathcal{G})$ the set of directed edges of $\mathcal{G}$. If $(i, j) \in \mathcal{E}(\mathcal{G})$ we call vertex i *child* of vertex j *parent*, with the child-parent link further denoted $i \to j$.

The directed graph $\mathcal{G}$ is acyclic if there exists no sequence of edges

$$[e_k = (i_k, j_k) \in \mathcal{E}] \text{ for } k = 1, \cdots K$$

such that

$$i_1 \to j_1 = i_2 \to j_2 = i_3 \to \cdots \to j_K = i_1.$$

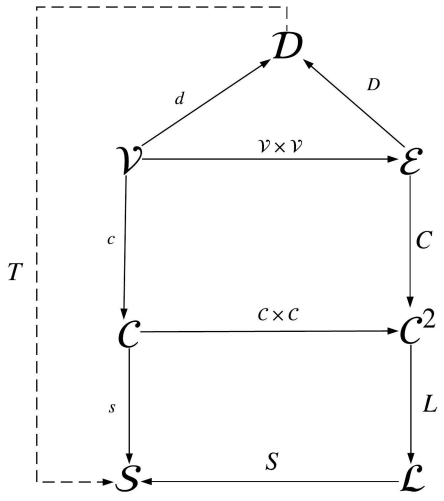


FIGURE 1. Spaces and Operators used to map the colored DAG to Schemas and Data

The directed graph $\mathcal{G}$ is colored if there is a mapping $C : \mathcal{V} \to \mathcal{C}$ where $\mathcal{C}$ denotes a finite set of classes of vertices. The traditional notion of a graph coloring seeks to assign the mapping $c(\mathcal{V})$ such that $c(i) \neq c(j)$ for any edge $e = (i, j) \in \mathcal{E}$. The following work deviates from this classical assumption, instead allowing the coloring to serve as an indication of intrinsic heterogeneity of the vertices. There is therefore a well defined set of edge classes $\mathcal{C}^2 = \mathcal{C} \times \mathcal{C}$ for which we extent the use of the operator $C : \mathcal{E} \to \mathcal{C}^2$. For any $e = (i, j) \in \mathcal{E}$, the edge class is $(c(i), c(j)) \in \mathcal{C}^2$ and the order matters. Due to the departure from the strict coloring definition there exists edges of type $(c, c) \in \mathcal{C}^2$.

2.2. **Coloring and Meta-Data Schemas.** Vertices represent abstract data about events or objects; this data has no assumed structure and is simply denoted $d(i) \in \mathcal{D}$ for any $i \in \mathcal{V}$. Without loss of generality, the set $\mathcal{D}$ is any data that can be expressed as a length l sequence of bits $\{0, 1\}^l$. In addition to the raw data $d(i)$ associated with vertex i, there is a meta-data schema s according to the color of $c(i)$.

For each $c \in \mathcal{C}$ define a meta-data schema $s(c) \in \mathcal{S}$; without loss of generality the schema set $\mathcal{S}$ denotes all possible logical data models Alan Chmura, J. Mark Heumann (2005). To ensure completeness of this abstraction, define color $c_0 \in \mathcal{C}$ as the class of vertex with null schema $s(c_0) = \emptyset$. Any vertex i such that $c(i) = c_0$ is simply a file stored with no structural meta-data declared[2].

The Colored DAG construction included the set of edge classes $\mathcal{C}^2$. It is prudent to extent our meta-data definition to include schema assignments for edges. In order to handle the

[2]The authors propose the convention that the null class c_0 be associated with the color black to denote opacity as compared to colors which provide some measure of information about the associated data object's contents

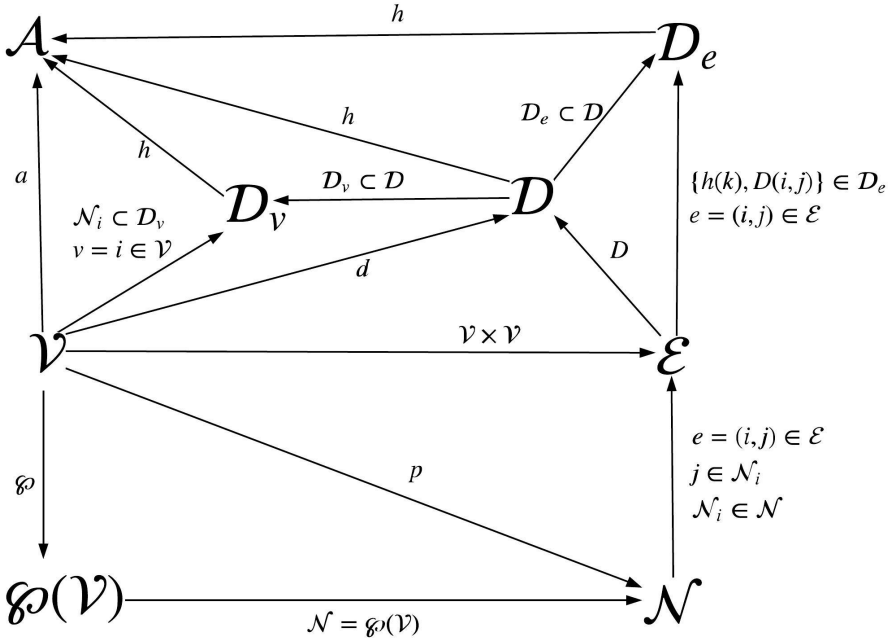


FIGURE 2. Spaces and Operators used to map the colored DAG to unique Content Addresses

general case, it is not assumed that there is a one-to-one mapping between elements of the set $\mathcal{C}^2$ and the edge coloring. Instead, denote the set of edge classes to $\mathcal{L}$ which is a finite set representing the domain of a mapping $L : \mathcal{E} \to \mathcal{L}$. In order to limit complexity, addition structure may be imposed such that for any edge class $C \in \mathcal{C}^2$ there is a subset of $\mathcal{L}_C \subseteq \mathcal{L}$ which is non-empty because necessarily contains the class C_0 associated with the null schema. Thus, for any edge $e = (i, j) \in \mathcal{E}$ there is an edge color $l(e) \in \mathcal{L} \subseteq \mathcal{L}_{C_e}$ where $C_e = (c(i), c(j)) \subseteq \mathcal{C}^2$. The edge color uniquely defines the edge schema according to the mapping $S : \mathcal{L} \to \mathcal{S}$. Metadata schemas are a generalized space such that the range of the mappings s taking vertex colors to schemas and the mapping S taking edge colors to schemas are the same.

To complete the formal function spaces, return to the notion $d(i) \in \mathcal{D}$ and observe that since $\mathcal{D}$ has no assumptions about structure it is suitable to define edge dimensional data $D(e) \in \mathcal{D}$ as well. The data can be described as having a schema in $\mathcal{S}$ irrespective of whether any addition meta-data is defined because the null schema is a member of $\mathcal{S}$. Naturally, any $d \in \mathcal{D}$ with non-null schema is expected to include a header containing its meta-data including its vertex or edge class. Therefore, the construct can be considered strongly typed by introducing an additional mapping $T : \mathcal{D} \to \mathcal{S}$. The operator T is presented with a dashed line in Figure 1 as its inclusion is optional.

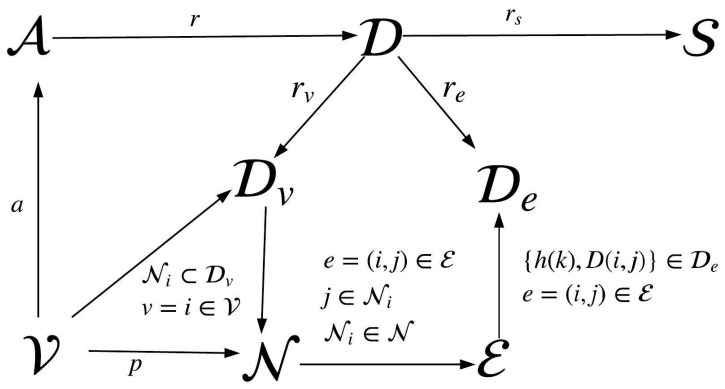


FIGURE 3. Spaces and Operators used to query data from vertices and edges in the colored DAG.

2.3. **Content-Addressability, Hash Functions and Edge Data.** The proposed Colored Directed Acyclic Graph is made up of vertices $i \in \mathcal{V}$ and edges $e = \mathcal{E}$. Let us define the address of any peice of data to be a cryptographic hash of the data, $h : \mathcal{D} \to \mathcal{A}$ where $\mathcal{A}$ is the range of the chosen hash function h. Applying the hash function h to the $d(i)$ for any vertex provides the address $a(i) = h\left(d(i)\right)$ of vertex i. In the proposed formalism the directed edges relating vertices to parent vertices are part of the vertex meta data.

A vertex is part of the Colored Directed Acyclic Graph if it has at least one edge. Without loss of generality, continue with the assumption that the graph $\mathcal{G}$ is a weakly connected digraph. Given this assumption every vertex in $\mathcal{V}$ must have at least one edge. The direction of the digraph is considered to flow from children to parents, therefore, vertices i with no parents are considered *leaves* and must have at least one child vertex in order to be connected.

Consider any vertex i that is not a leaf, that has color $c_i \in \mathcal{C}$ with schema $s(c_i)$ which includes a mapping to its parent vertices, $p : \mathcal{V} \to \mathcal{N}$ where $\mathcal{N} = \wp(\mathcal{V})$ is the set of all possible neighborhoods, denoted by powerset of vertices $\wp(\mathcal{V})$. Note that this simply recovers the fact that a vertex may be connected to any other vertex: $p(i) = [1, \ldots, K] = \mathcal{N}_i$ where $\mathcal{N}_i \in \mathcal{N}$ is a neighborhood (or parent set). That is to say: $k \in p(i)$ if and only if $(i, k) \in \mathcal{E}$. The data of node i then necessarily contains the data associated with each of its edges $D(e)$ for each e.

It is prudent define the data $d(i) = \{d_0(i), d_{e_1}(i), \ldots, d_{e_K}(i)\}$ where $d_0(i)$ is the data for the vertex i itself and $d_{(i,k)}(i) = (a(k), D(i, k))$ for all vertices k with addresses $a(k) = h(d(k))$ which are declared parents of i at the time i created. The edge classes and associated schemas for edges are used to ensure the data $D(i, k)$ collected from within $d(i)$ are defined on an edge by edge basis according to the formalisms in section 2.2.

2.4. **Reading Data From Content Networks.** So far only the mappings for storing data have been defined. In order to effectively use a content network one must also define the functions that read the stored data. The basic query operation for the network is denote $r : \mathcal{A} \to \mathcal{D}$. The mapping takes the hash representing the address of the data and returns the data itself. This top level query operation returns and object that without further qualification has no implied structure. However, by following a coloring scheme that includes a metadata header with information about the data's schema, it is possible to make the data contained a more generally useful domain for other operators.

Consider, three addition types of read operations for any particular object $d \in \mathcal{D}$, read structure $r_s : \mathcal{D} \to \mathcal{S}$, read vertex $r_v : \mathcal{D} \to \mathcal{D}_v$ and $r_e : \mathcal{D} \to \mathcal{D}_e$. Note that it may be necessary to apply r_s in order to get the necessary structural information about $d \in \mathcal{D}$ required to actually apply r_v and that furthermore, r_v returns $d \in \mathcal{D}_v$ which will include the neighborhood or parent set information required to identify and query edges using r_e.

Therefore, querying data from the network may require a sequence of calls, the imposed structure guarantees that all data about vertices and edges is recoverable from a nested data using well defined content hash addresses; that is any data object $d \in \mathcal{D}_v \cup \mathcal{D}_e \subseteq \mathcal{D}$ has a content defined address $h(d)$. Note that the mapping $T : \mathcal{D} \to \mathcal{S}$ proposed in section 2.2 used to impose typing on data is equivalent to r_s if it is included. If strong typing is not implemented then r_s still exists but its range is any data $\mathcal{S} = \mathcal{D}$ rather than some more narrowly defined set of supported predefined structures or schemas.

2.5. **Partial Orderings and Directed Root Trees.** In sections 2.2, 2.3 and 2.4 the definitions for a network of content with color based classification and parent child relationships have been established. In this section, the network structure is exploited to define useful sub-graphs, specifically directed rooted trees with proscribed selection rules based on class data for vertices and edges.

First observe that there is a partial ordering over all vertices: let $i < j$ if and only if there exists a directed path

$$i \to k_1 \to k_2 \to \cdots \to j.$$

Therefore, for any vertex i there is a set of ancestors $\{j \in \mathcal{V} \,|\, j > i\}$. This set can be constructed iteratively by backpropogating through the local parent sets $p(k) = \mathcal{N}_k$ for all vertices k starting with the immediate parent set $p(i) = \mathcal{N}_i$ and continuing until only leaf vertices remain. The resulting network is a directed root tree with edges pointed toward the root vertex i.

An important property of the vertex coloring $c(i)$ and edge coloring $L(e)$ classes is that these directed root trees may be further restricted based on the class of data they contain or the nature of the link from parent to child. Define a colored directed root tree to be

$$\mathcal{G}_{tree} = \{\mathcal{V}_{tree}, \mathcal{E}_{tree}\} \subset \mathcal{G}$$

such that $\mathcal{V}_{tree} = \{i \in \mathcal{V} \,|\, c(i) \in \mathcal{C}_{tree}\}$ and $\mathcal{E}_{tree} = \{e = (i,j) \in \mathcal{V} \,|\, L(i,j) \in \mathcal{L}_{tree}\}$.

2.6. **Attribution Metrics Over Colored Trees.** By including only vertex and edges whose classes contain structure it is possible to apply attribution algorithms over a well defined domain. Furthermore, it is even possible to dynamically define the backpropogation

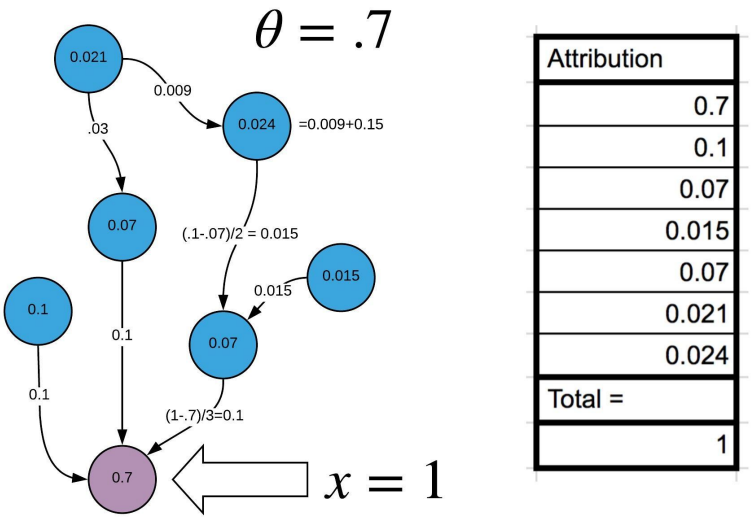


FIGURE 4. Simple Example of a Backpropogation $F(x)$ with self-weight $\theta = 0.7$

algorithm in terms of the data discovered in the nodes, i.e. explore backwards through vertices of class $c_1, c_2, c_3 \in \mathcal{C}$, stop at vertices of class $c_4 \in \mathcal{C}$ and apply some function $f(d)$ that is well defined for data d with structure $s(c_4)$.

Let us define a general class of attribution metrics

$$F : \mathbb{R}_+ \longrightarrow \mathbb{R}_+ \times \mathcal{G}$$

which can take value $x \in \mathbb{R}_+$ assigned to any vertex $i \in \mathcal{V}$ in the the colored directed acyclic graph, using any arbitrary backpropogation rule $B : \mathcal{V} \to \mathcal{G}_{tree} \subseteq \mathcal{G}$ and assign a value $x_k \in \mathbb{R}_+$ to each vertex $k \in \mathcal{V}_{tree}$ and to each edge $e \in \mathcal{E}_{tree}$ to the contribution of the discovered ancestors according to the vertex and edge data along the path.

Further observe that while it may or may not be a desirable property, it is strictly feasible to implement algorithms which are conservative in one or more dimensions by allocating a percentage of remaining attribution as one backpropogates. Consider an algorithm that attributes a $\theta \in (0, 1)$ share to each vertex k traversed and $\frac{(1-\theta)}{|\mathcal{N}_k|}$ to each parent vertex until the leaves are reached or the remaining value falls below some threshold ϵ at which point all remaining value is attributed. Under algorithms of this type, it is guaranteed that

$$x = \sum_{i \in \mathcal{V}} F_i(x).$$

This is merely an example meant to show that the colored directed acyclic graph with defined color based classes of data can serve as the domain for a large class of algorithms that suite the needs of the algorithm designer. This concept will be further explored in more concrete cases.

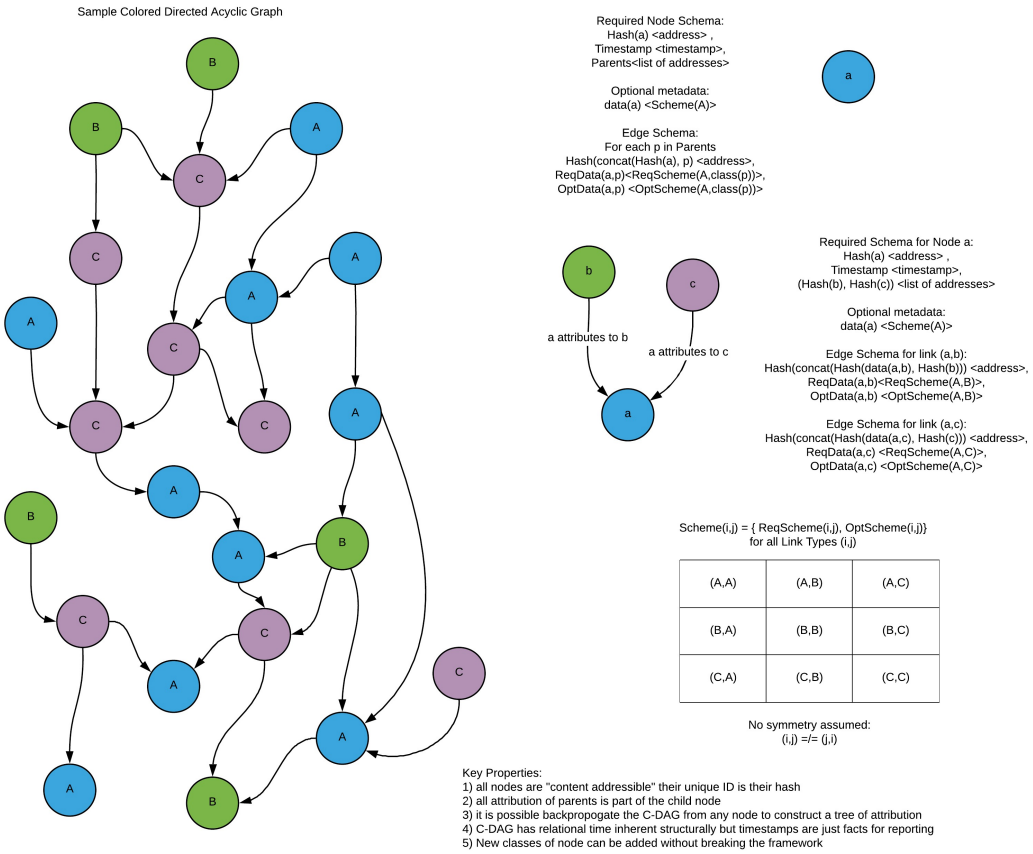


FIGURE 5. Needs to be reworked slightly match text

3. Multiclass Attribution Networks

In this section, a multiclass attribution network is constructed and protocols for storing and querying data from the network are defined. Consider a multiclass attribution network with vertex classes $\mathcal{C} = \{c_1, c_2, c_3\}$ and edge classes

$$\mathcal{C}^2 \subseteq \{(c_1, c_1), (c_1, c_2), (c_1, c_3), (c_2, c_1), (c_2, c_2), \ldots, (c_3, c_3)\}.$$

For simplicity, let $\mathcal{L} = \mathcal{C}^2$ indicating the case that there is only one type of link class for each vertex class pair. Furthermore, assume that for each class c the there is a metadata

header after the following fashion:

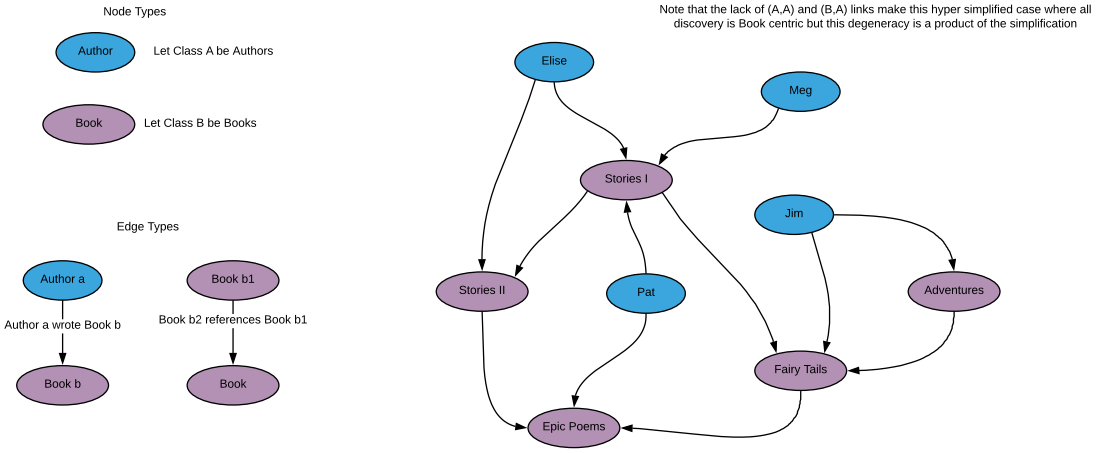
$$\begin{aligned} <address>: \{&'Title' :< str >, \\ &'Account' :< address >, \\ &'Software' :< address >, \\ &'Datetime' :< timestamp >, \\ &'Class' :< address >, \\ &'Parents' : \{< address >:< struct >\}, \\ &'Data' :< struct >\}. \end{aligned}$$

Let $c_1 =' Account'$ denote identities within the network, $c_2 =' Software'$ be vertices containing data which is code, $c_3 =' Other'$ be any other files stored in the network. Account and Software appear as metadata fields because every object in the network must be written by some account using some software regardless of which class the object itself is.

3.1. **Account Class.** For this network accounts will be important but simplest objects defined by a public key, private key pair with its metadata containing its Class, the string 'Account' and hash of the software used to created the account proven to be stored at the address matching its hash. In the event that an account is created by a previously unregistered piece of code, a new code object will be stored as a subroutine.

3.2. **Software Class.** Software may also be a simple object but its metadata will also allow for explicit declarations of parents of Classes 'Account', 'Software' or 'Other'. Declaring an account as a parent is different from being the account which signed the transaction to the network; a parent 'Account' is a citation of some kind which may represent authorship, sponsorship or another relationship depending on the edge metadata supplied in the 'Parents' Structure. Likewise, a parent 'Software' may be a dependency, or forked version of software already stored in the network.

3.3. **Class Class.** The 'Class' class is an object encoding the metadata for a class. Its metadata will also allow for explicit declarations of parents of Classes 'Account', 'Software' or 'Other'. Declaring Class as a parent is only well defined for other 'Class' objects and implies inheritance; a parent 'Account' of a 'Class' is a citation of some kind which may represent authorship, sponsorship or another relationship depending on the edge metadata supplied in the 'Parents' Structure. It is important for it to be possible to create new classes both to support new use cases as well as to extend or evolve existing use cases. The class 'Class' along with 'Accounts' and 'Software' provide the backbone of the attribution network with explicit dependencies on whereas the parents are implicit assignments of credit to other pre-existing objects in the network.



Books	Address	Datetime Published	Parents Name	Parents Address	Link Address	Parent Type
Stories I	b1f82c02f194f875f61b353524b11822	8/30/2017 3:55:20	Elise	182cd31f91d95d4815d32d8906469a48	4833ad8271f6b7d8af67afc419afc581	Author
			Meg	b8bc42495234cdcdcd3ed49dad120a84	48183ab20f0b6f9b746c84109a8fafab	Author
			Pat	532762fa5a5b7169aa4dd24717ba9df9	b816cd7103abc18eb10bfbe8e70dced5	Author
Adventures	515a3f4a3ef00683abbd16ab9abfb94e	3/2/2018 1:01:01	Jim	d54b3c8fcd5ba07e47b400e69a287966	1e6719813f7bc5736abd24947a5ba1e2	Author
Stories II	3d16d438f8cb1bd5e4c9a863c3c0ec2e	5/30/2018 3:03:34	Elise	182cd31f91d95d4815d32d8906469a48	06e5c8f724ae05a60c183f98aa866cea	Author
			Stories I	b1f82c02f194f875f61b353524b11822	a557a1aa70f8fec9d71a2dc5bff913f1	Book
FairyTails	6d018bf383033b5636430428bd4b6db3	6/30/2018 11:11:01	Jim	d54b3c8fcd5ba07e47b400e69a287966	b5866630ca21f397088a9d6029ff97c1	Author
			Stories I	b1f82c02f194f875f61b353524b11822	a557a1aa70f8fec9d71a2dc5bff913f1	Book
			Adventures	515a3f4a3ef00683abbd16ab9abfb94e	164863d85d0a60b3d06e3a313bf668a8	Book
Epic Poems	a72e4f95d18c06753ee9fbc8e4ead24d	9/10/2018 12:00:00	Pat	532762fa5a5b7169aa4dd24717ba9df9	16d18d416ddd3de0bb6c939bd964bfc4	Author
			Adventures	515a3f4a3ef00683abbd16ab9abfb94e	164863d85d0a60b3d06e3a313bf668a8	Book
			Stories II	3d16d438f8cb1bd5e4c9a863c3c0ec2e	936364bfaa57ec1760342a00d9af9693	Book

Author	Address	datetime Joined	~~Parents Name~~	~~Parents~~	~~Link Address~~	~~Parent Type~~
Elise	182cd31f91d95d4815d32d8906469a48	8/3/2016 1:00:01	Authors aren't equipped with inbound links in this simple example			
Meg	b8bc42495234cdcdcd3ed49dad120a84	8/14/2016 1:00:00				
Jim	d54b3c8fcd5ba07e47b400e69a287966	8/26/2016 7:07:07				
Pat	532762fa5a5b7169aa4dd24717ba9df9	8/22/2016 3:10:10				

FIGURE 6. Z's Notes super simple example of made of data

3.4. **Other Class.** The class 'Other' is a placeholder class which can be used to contain any number of richer classes of data with well defined metadata headers. For the purpose of this example network all addition classes lumped into other have metadata including 'Account' and 'Software' which must be references to existing objects or those objects will be generated and stored as part of the process of creating the new object of class 'Other'. Additionally, it is assumed that there may be a wide variety of parent relationships associated with the 'Other' class but the proposed network is sufficiently general for the implementing team to define data models as needed over a wide array of additional classes.

3.5. **Pen Names and Sybil Attacks.** Describe the Books and Authors example where Authors fill the role of Account and for simplicity we neglect the software class. The general applicability of the framework to the structure of real life questions of attribution is demonstrated, including the implications of accounts themselves not having parents. While it is possible to generate many accounts and thus lack a single identity in this framework,

one would argue that attribution in such a network would be a reward not a cost and fragmenting that attribution across multiple identities would provide no additional utility in most cases. The book example does however bring to mind the case of pen names, so it is plausible that an author might choose to have different accounts for personal reasons. Due to the overall flexibility of this network as a domain over which incentive algorithms may be defined, the decision regarding counting attribution for any particular algorithm determine the impact of Sybil attacks on that algorithms output.

4. Application: Knowledge, Science, and Open Innovation

Introducing a notion of incentives and intent can help demonstrate a powerful use case of these multi-class attribution networks: a Directed Acyclic Memex (DAM.) Consider the example of Planck, an incentive system for encouraging open innovation and knowledge sharing. In this system, ideas are encoded as Glyphs— digital blockchain objects which, for present purposes, can be thought of as vertices in a DAG. A new Author's Glyph, called a Glyph_A contains a content address, some parent addresses, and at least one unique value (a uint256, for instance.) A Glyph_A may cite other Glyphs as "parent addresses", so for simplicity we'll refer to these addresses as Glyph_{cited}. When a prize is awarded Glyph_A, some proportion of the prize can go to the Glyph_{cited}. Each of those Glyph_{cited}, as they get their proportion of the prize, may also allocate some proportion to the Glyphs *they* cite.

You can visualize this as an irrigation system, with prize compensation flowing initially to the Glyph_A which, like a Dam, has "gates" allocating flow to specific Glyphs_{cited}. Each Glyph_{cited} is a catchment for some funds, and also has gates of its own, allocating funds to the Glyph_{cited}. Eventually, the water fails to fill a catchment and the flow for that prize stops. Figure 6 presents the visual analogy comparing Planck's use of a DAM to an actual Dam.

The awarded Glyph_A acts as both a catchment for some amount of funding and a pre-set system of sluices which direct the remaining proportion.

4.1. Designing Incentives.

The application of any abstract model to a human system presents a special problem, popularly known in economics as the "Lucas Critique". The Lucas Critique, loosely speaking, says that any model of the human world is going to exist in the human world, which may then react to the model.[3] We will demonstrate that drawbacks to a naive application of such a DAG to the incentive design described in the previous section, in which the children of a vertex are rewarded according to a revenue share such $A = \theta R$, with $0 < \theta < 1$, and the remainder going to attributed vertices according to some function $f : \mathbb{R}_+ \times \mathcal{D}^m \to \mathbb{R}_+^m$.

If we imagine the production of knowledge as a cooperative game, we might imagine a reasonable functional form would be the Shapley Value for each vertex, as follows:

$$\phi_i(G) = \frac{1}{\mid N \mid !} \sum_{\pi \in \Pi_N} \Delta_\pi^G(i)$$

[3]Threat heuristics like "Sybil Attacks" are guided by a similar logic.

Where G is a characteristic function game, and the function defines weighted payoff vectors according to the marginal contribution of each player i. And yet, the true marginal contribution is rarely available. With respect to attribution, for instance, each player has the incentive to maximize their claim and over-report their marginal contribution. This is why we mentioned the Lucas Critique: while existing scientific citations may be very well-modeled with a DAG, any incentive game will change the incentives for attribution, perhaps for the worse. To state an extreme but simple case: if the claimant of Vertex A gets to choose θ, their rational choice is $\theta = 1$.

Nodes:

- ID: Contributors (human identity),
- G: Grants (financial support)
- R: Research Papers (public natural text, figures, etc)
- C: Open Source Projects (public code - assuming git conventions)

Edges:

- (G,ID) id puts money in grant
- (R,ID) id authors research paper
- (C,ID) id contributes to open source project
- (R, G) g funds R
- (C, G) g funds C
- ...
- you get the idea: you need to define all the types of links you need and what their ReqSchemes and OptSchemes are to actually establish a protocol

Incentive Game: Define a really simple one, show it fits the back prop model (its an example not a proposal)

5. Conclusion

math tools for making real things! Your Open Science project looking for open source coder type collaborators right? Say so. Reference related projects in the space researchers.one, etc.

Citations but not overkill.