Bewilder: Handling Web Resource Complexity in Online Learning and Research

Eric Easthope*

Department of Computer Science, University of British Columbia

ABSTRACT

Bewilder is an online tool for annotating, re-arranging, and connecting web links in a partially hierarchical way based on intrinsic and retrieved web link metadata. Bewilder makes saved web link metadata editable, and derives connections from web link metadata similarities. Connections are emphasized through various default or user-specified undirected and directed visual marks, and identified through user interactions. Users can also download modified web link metadata.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Graph drawings; Human-centered computing—Visualization—Visualization design and evaluation methods; Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Web-based interaction;

1 INTRODUCTION

Organizing web links like Uniform Resource Identifiers (URIs), Uniform Resource Locators (URLs, a subset of URIs), and Document Object Identifiers (DOIs), which is often done within browser-based folders, is still a visually cluttered and awkward user experience. The alternative, to save and archive tabs offline, again usually within folders, is arguably no more effective. Given many web items, the amount of manual categorization a user must do quickly becomes overwhelming. Also, there is no recall for *why*, or even *when* web links have been saved in the order they have. In the context of saving too many in-browser tabs, this accumulation of unordered web items has sometimes been colloquially called "tab overload", or more evocatively "tab overload hell."

Many in-browser and standalone bookmark and resource managers put heavy emphasis on users to manually annotate and arrange their own content without much automation. Some newer in-browser bookmark features, like "tab grouping," which lets a user arbitrarily aggregate tabs, do help users become more organized, but are often in effect just different views of folders or other tree-like data structures.

The complexity of these tasks increases further if a user also wants to organize and connect common web URLs, such as those stored by tabs, to other URI/DOI-based resources such as arXiv papers, for example the DOI `arXiv:xxxx.xxxx`, which corresponds to the URL `https://arxiv.org/abs/xxxx.xxxx`, and also government information. A user may also wish to add notes and other annotations to web link metadata to recall meaningful information later about why they have saved web links in the order they are in, and how the web links are connected. Two key visual challenges arise:

1. How do we make it visually and cognitively easier to collect, review, and share web links in a number of different contexts?
2. How do we visualize these collections at scale?

Bewilder is a software tool that gives users a way to overview, annotate, and arrange collected web items in a way that retains *why* and *when* we are saving them, and how they are connected to other resources. This tool replaces tab-based overload with interactive visualizations of saved web links and their connections through similarities in their metadata. Bewilder also generates portable representations of how these URL-like resources are related to each other to facilitate and support large-scale browser-heavy research, learning, journalism, and so on. Our design of Bewilder is rooted in a desire to give users more control over web link and research article collections, as well as a way to modify, explore, and enjoy automatically generated arrangements of them.

1.1 Personal Expertise

How to construct effective representations of connected web link and scraped webpage data, and how these representations could support learning and research, has been an intermittent interest over the past few years, but without much external feedback or expert knowledge. Previous efforts focused more on the programmatic retrieval and processing of online content, but not so much the visualization of this content, nor the integration of other user-collected data. In preparation for this work, we have engaged with learners and researchers directly to learn about what tools they currently use to archive and retrieve online content, and some the problems they encounter within their online workflows.

We have also prototyped various software tools in the past using D3.js with the broader intent to support our own data-driven work and research. Some of these previous developments also centred on the problem of digital notetaking and how to connect, share, and retrieve learning resources without making full copies of them.

2 RELATED WORK

Web link archiving and annotation have been addressed by numerous commercial software tools. There are many bookmark managers like *Raindrop.io* [12], *Pocket* [11], and *Dropmark* [14], amongst others, and many of them also have sharing capabilities. *Zotero* [17] is popular too, particularly for managing complex research-driven bibliographies.

Zettelkasten [6] is a knowledge management system that was popularized by sociologist Niklas Luhmann that regards physical collections of notes as hyperlinked documents. A software tool called *The Archive* [21] makes this system digital, and there are other software derivatives such as *Roam* [16], *TiddlyWiki* [23], and more recently *Obsidian* [15]. Notes taken within these tools can also include web links. Also, for research there is *StArt* [4], a systematic literature review and mapping tool, which also produces links between large numbers of content-rich documents.

In terms of large-scale online collections, there is the *Open Graph Protocol* [5], the *Google Knowledge Graph* [19], as well as the *Dataverse Project* [10]. The Open Graph Protocol turns web pages into rich data objects in a social and sharing context through attributes such as website titles, types, URLs, and images, as well as other optional metadata. The Google Knowledge Graph is a database of

*e-mail: eric.easthope@me.com

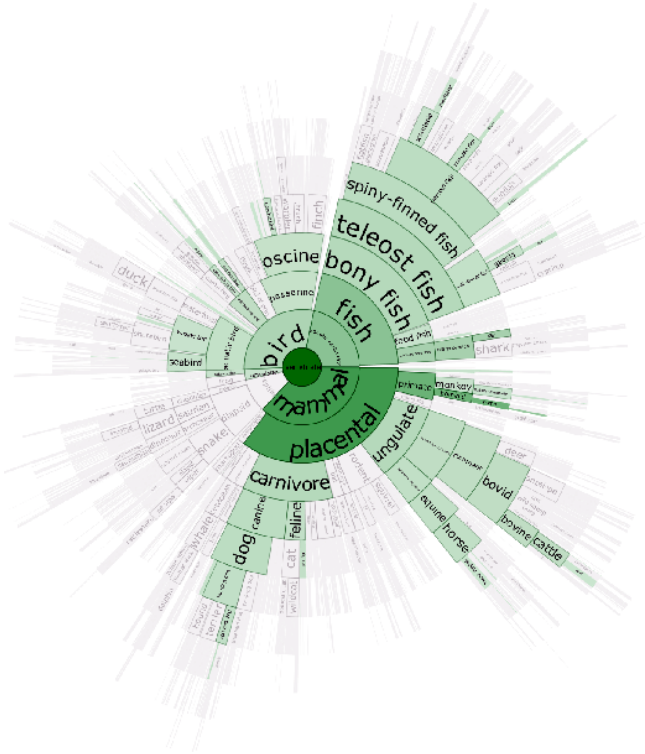

Figure 1: A radial hierarchical view in Docuburst [3].

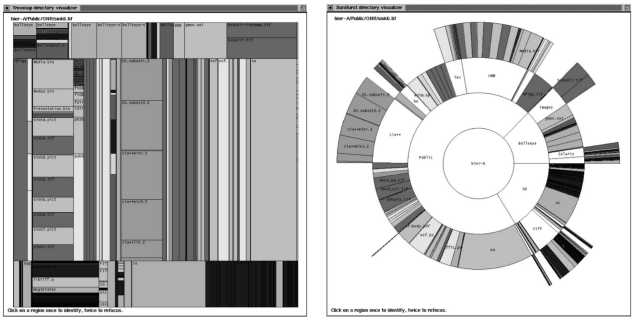

Figure 2: Comparison of rectangular treemap and Sunburst views of a file hierarchy [20].

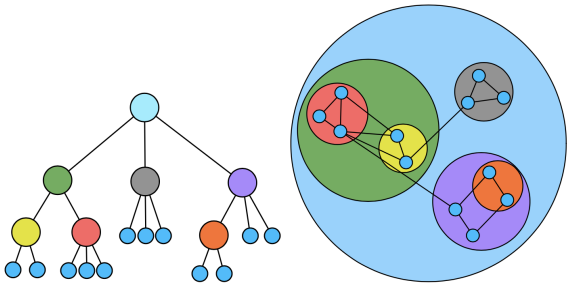

Figure 3: Concentric circles show hierarchy through containment in GrouseFlocks [1].

collected information used to enhance search results. The Dataverse Project provides open source and shareable collections of research data. The existence of these collections seems to reflect a public desire for interlinked and metadata-rich web content. Also, Tumpa and Masroor Ali have shown that document concept hierarchies can be derived from structures similar to the Google Knowledge Graph [22].

In terms of visualizing connections between interlinked web content, the "discussion topology" view in *Kialo* [2], shown in Fig. 4, is an example of visualizing a radial tree-like topology. In particular, Kialo visualizes hierarchies of arguments between participants on a topic.

Radial visualizations like these are also seen in *Docuburst* [3], which hierarchically visualizes document content based on language, as well as in a literature review on visualizing ontologies [18], which discusses the utility of numerous hierarchical radial layouts for representing ordered relationships between items. Docuburst is shown in Fig. 1. Many of these visualizations seem to be derived from the *Sunburst* radial visualization technique developed in 2000 by Stasko et al., demonstrating its applicability to a number of diverse problem domains. Stasko et al. found that participants preferred Sunburst views over rectangular treemaps, which are also used to display hierarchical data, and that participants felt that Sunburst views conveyed structures and hierarchies better [20]. See Fig. 2 for a direct comparison of a rectangular treemap and Sunburst.

GrouseFlocks [1], shown in Fig. 3, is another example of visualizing tree-like hierarchies derived from graphs. GrouseFlocks uses containment within concentric circles to represent these hierarchies.

Friedrich and Schreiber, as well as North and Woodhull, have also explored layouts for hierarchical graphs to account for nodes of arbitrary sizes arranged upon multiple hierarchical levels [7, 13]. Finally, Hong and Nikolov have explored hierarchical graph layouts in three dimensions [9].

Something that underlies many of these online learning and research tools is a lack of user and data interactivity, and often a lack of visualization entirely. In particular, the underlying topologies of the Open Graph Protocol and the Google Knowledge Graph could reveal interesting connections if they were visible. However, neither dataset is readily accessible to the public, let alone conveniently available to end users. Data that is hosted by the Dataverse Project is publicly accessible, but there is still a lack of dedicated visualization tools.

Another pitfall of many of these derived software tools and visualizations, such as those developed for *Zettelkasten*, is that users cannot navigate nor edit hierarchical content directly. These tools often make users move between multiple user interface (UI) views. For example, the Kialo UI makes users contribute arguments indirectly through text-based subpages, shown in Fig. 5, instead of directly through the "discussion topology" view. This adds unnecessary cognitive load by requiring users to navigate through these nested text-based subpages.

Overall, we think that gaps such as these in user and data interactivity remove user freedom to do more important underlying tasks, such as recognizing and even acting upon meaningful connections between items. We also think that users should be able to impose their own orderings on collections to reinterpret and better understand them in a personal way.

3 Data

To give users this control over collections of web links and research articles, especially to modify, arrange, and explore details of these collections, Bewilder unifies a number of often disparate forms of hyperlinked online web content. This web content is frequently saved as `.webloc`, `.url`, and HTML (Hypertext Markup Language) files. Both `.webloc` and `.url` files store a single URL. Many modern web browsers also provide a bookmark export feature to produce HTML-based archives of web links, and often store both web URLs

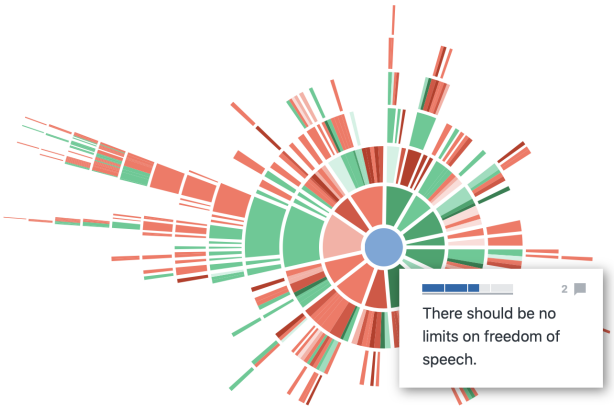

Figure 4: The “discussion topology” view in Kialo [2].

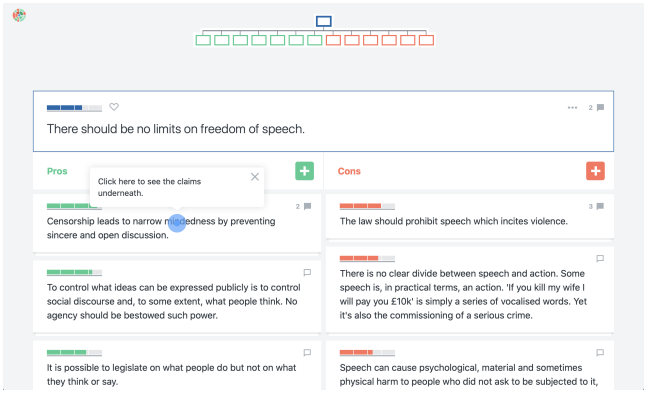

Figure 5: A text-based subpage to contribute arguments to Kialo [2].

and titles of individual webpages. These exported bookmarks are structured as nested hierarchies of web links, where HTML headers, subheaders, and so on, represent user-customized bookmark folders.

All of these file formats, that is `.webloc`, `.url`, and HTML, can be unified as nested JavaScript Object Notation (JSON) data. Also, we can scrape webpage metadata such as social and sharing data using the Open Graph Protocol. Some of this metadata is shown in Fig. 6. Scraping this metadata gives us programmatic access to titles, types, such as `video.movie`, images associated to each web link, and more. Many DOIs are also directly associable to specific URLs, such as the arXiv paper example before, and can therefore be represented within JSON.

Once we have these aggregate collections of user-provided web links from HTML, `.url`, or otherwise, combined with additional social and sharing content from the Open Graph Protocol, we can make simple comparisons of URLs, timestamps, user-specified categories, and more, to derive a JSON object containing node and link structures, and their attributes. Here, nodes represent web items, and links represent similarities in their metadata.

Each of these nodes contains its own associated metadata, and this metadata is a sequence of key-value pairs for each node attribute, such as `title`, `url`, `categories`, `notes`, etc. In application, this sequence of key-value pairs forms a JavaScript object that can be nested and referenced within other JavaScript objects. This data structure makes it easy to identify a node by its unique identifier, and modify its metadata by mutating key-value pairs. Also, since web links are often nested within subheaders in the conventional HTML bookmark file format, each node also contains a `level` attribute to represent hierarchical ordering.

Figure 6: Sample of node metadata retrieved from the Open Graph Protocol [5], and links derived from similar node metadata.

Links, which are also sequences of key-value pairs specifying which two nodes they connect, as well as their directionality, can be created or updated if metadata similarities are found between nodes. More connections can be made if there are similarities in the retrieved Open Graph Protocol metadata of two nodes.

These data representations, coupled with the task abstraction that follows, define the outcomes for Bewilder.

4 Abstraction

Bewilder is designed around the idea of “user stories,” which are generalizations derived from discussions and occasional but brief interactions had with various students, researchers, and other learners and information foragers outside of the academic community. Discussions usually involved comparisons of personal learning approaches and/or what media people used for their notetaking and research. One graduate student provided a sample of one hour of their online research notes, which contained more than 70 web links in between loosely strewn tags, images, and comments. We also have a number of personal bookmark collections exported as HTML from the *Google Chrome* and *Safari* web browsers, as well as bookmarks from Raindrop.io [12], which span a few years of archiving.

To support collections like these, Bewilder produces graphically scalable overviews for up to one hundred or more annotated and interdependent node-like web items. Bewilder also exposes web link metadata, such as the titles, types, and images mentioned, and lets users annotate nodes through lists of categories, and hierarchical attributes, to derive new dependencies between web items and accordingly new overall web collection topologies.

Each user story that we define briefly describes a practical use case for Bewilder:

1. "*As a student I want to be able to quickly aggregate and group web links by their similarities so that I can solve resource-heavy assignments faster.*"

2. "*As a researcher I want to be able to re-arrange and annotate web links so that I can take notes on a per-link basis during literature search.*"

3. "*As a journalist I want to be able to make directed and perhaps causal ties between web links so that I can trace through the events leading up to a breaking story.*"

4. "*As a self-directed learner I want to be able to create and annotate web links with traceback so that I can recall later how I found specific information.*"

All four users are trying to **derive** *tree* or *graph*-like data from individual web *items*. Recall that web items are represented by *nodes*, and their connections, derived from similarities in node metadata, for example similar categorical types or overlapping categories, are represented by *links*. Hierarchical attributes can be used to derive a directed graph representation of these web items.

User (1) takes this directed node-link data, and combines it with categorical node *attributes*, either annotated by the user or retrieved from Open Graph Protocol [5] metadata, to derive a new graph *topology* by adding links. These links indicate dependencies, in particular similarities, of attributes within the graph. The user can then **reduce** this graph by *aggregation* based on these dependencies.

User (2), like User (1), annotates nodes within a graph to add categorical attributes to produce new node-to-node dependencies. User (2) can then **encode** new graph arrangements, *ordered* or otherwise, to **identify** *features* and/or *outliers*.

Users (3) and (4) perform mutually similar tasks. However, unlike Users (1) and (2), these users produce and **arrange** a directed graph topology by adding many nodes, then adding categorical attributes to nodes to derive dependencies, which produces links, and finally reordering these nodes based on the order in which they are added, which is derived from a `datetime` attribute or otherwise.

These user scenarios inform the functional design of Bewilder.

5 Solution

Bewilder combines a radial partially ordered hierarchy, shown in Fig. 8 as annuli, each with a subsequently larger radius, and a force-directed node-link view, to create a sense of interdependence and hierarchy between web items, which are nodes. Each annulus represents a different `level` attribute value and the largest annulus corresponds to the maximum `level` attribute value amongst the web items shown. Additional or fewer annuli are automatically rendered by increasing or reducing this maximum value, respectively. Also, to emphasize each level, every annulus is coloured using an interpolated greyscale colour map. These particular greyscale values are chosen heuristically.

There are three buttons in the Bewilder UI, which are also shown in Fig. 8:

- The leftmost button toggles "default connections," which if activated, as it is by default, creates links marked by dashed lines between any two nodes if their base URL is identical. To clarify, the base URL of `https://example.com/subdirectory` is simply `https://example.com`. See Fig. 11 for an example of these "default connections." In Discussion & Future Work we discuss other possibilities for this control.

- The centermost button downloads the current JavaScript data object as a *minified* JSON file, meaning that all whitespace is removed.

Figure 7: D3.js-based purple-blue-green colour scale for nodes from `d3-scale-chromatic`.

- The rightmost button toggles "chronology," which is currently disabled. In Discussion & Future Work we discuss a possible development direction for this control.

Users can add web links by dragging in and dropping `.webloc`, `.url`, and HTML bookmark files from their file system. Upon dropping the file, Bewilder loads it, parses its contents, retrieves scraped Open Graph Protocol metadata, and produces a JSON-like data structure as a JavaScript object. HTML files in particular are parsed as JSON based on the nesting of web links within bookmark folders, which is expressed in HTML as web links under one or more subheaders. Newly added nodes are assigned a `level` attribute value by default, namely an integer, that corresponds to the outermost annulus.

From this JavaScript data object, links are automatically derived from similar node metadata attributes, as shown in Fig. 6. In particular, Bewilder creates undirected links if two nodes have at least one category in common with respect to the `categories` attribute of each node, or a directed link if one node is specified as a "source" of another node. Doing so produces tooltip-compatible information for every web item based on its metadata and also its interdependencies with other web items.

Nodes are shown as coloured circular marks using an interpolated purple-blue-green colour map, shown in Fig. 7, where the colour of each node is determined linearly from its `level` attribute value with reference to the zeroth level, that is the centre, which corresponds to green. Nodes further from the centre are then coloured more blue, and then more purple. The choice of three colours is to better emphasize perceptual differences between intermediate levels without going further and using rainbow colour maps. If the maximum `level` attribute value changes, all nodes are re-coloured so that the colour of nodes at the inner and outermost annuli remain fixed.

Where nodes are placed on the radial hierarchy is not strictly enforced, and any node can be dragged and dropped into another annulus. Dragging a node to another level updates the `level` attribute of that node.

If a node is specified as the "source" of another node, its radius is increased. This increased radius emphasizes connected or "central" nodes within the hierarchy. These are nodes that are not connected to "targets" or do not appear within `sources` attribute list of any other node. Technically speaking, these nodes without "targets" are called *leaves*.

Three text components appear if the cursor is placed, that is *hovered*, over a node. The left text component displays the titles of "source" nodes connected to the hovered over node, in particular the nodes listed in `sources`, and the right text component displays the titles of its connected "target" nodes. Each node is identified in `sources` with a Universally Unique Identifier (UUID), which is also its `id` attribute. The title of the hovered-over node, as well as its associated `categories` and `notes` attributes as shown in Fig. 11, are displayed below the radial view.

These directed connections between nodes are indicated by incoming and outgoing arrows in the graph view, where incoming arrows indicate "sources" and outgoing arrows indicate "targets." Once two nodes are connected by a categorical or directed link, any "default" links are replaced, giving categorical and directed links some precedence, as they are user-specified or derived from bookmark folder hierarchies.

Clicking and holding a node for a moment reveals an annotation view, shown in Fig. 9, which is a semi-opaque overlay on top of the

primary radial and graph view. The annotation view displays and allows a user to see or modify node attributes, such as its `categories`, which result in new undirected links if "default connections" is toggled, its `sources`, which results in new directed links, or its `notes`. The choice of making the annotation view semi-opaque is to allow users to retain a sense of visual context, while still focusing their attention on modifying node-specific attributes. Also, changes to the `categories` or `sources` attributes of a node which produce new links are often immediately rendered, so some opacity gives us visual feedback when the underlying graph changes. Clicking outside the annotation view hides it and returns the user to the primary view.

5.1 Implementation

Bewilder is written in JavaScript and hosted online at `www.bewilder.me` with free hosting from Vercel. Vercel provides a minimal back-end for scraping webpage Open Graph Protocol metadata, front-end hosting for the Bewilder UI, and enables continuous deployment from GitHub by automatically updating Bewilder when new production versions are released. Accordingly, source files for Bewilder are version-controlled and hosted in a private repository on GitHub.

All visualizations are also written in JavaScript using D3.js and the React.js framework, and some UI elements are server-side rendered through Next.js. Next.js renders Bewilder as a single-page application. D3.js is used for rendering the radial and force-directed graph view elements, for overall data manipulation, and for modifying the appearance of some functional UI elements such as the hover-activated text previews. React.js is used for UI layout and to develop component-based views. This view component modularity greatly assists with debugging and troubleshooting, and promotes code reusability. Redux is also used to manage UI-related state.

We considered using TypeScript to add simple checks for matching data types, and to catch and prevent unexpected data handling errors while importing web links, but using TypeScript did not seem to be necessary at this stage as we had sufficient checks for file extensions during file loading. We were also borrowing some personally authored code to get URLs from imported web link files. However, eventually this code was replaced to support asynchronous Open Graph Protocol metadata retrieval and multi-file loading.

5.2 Milestones

Here we summarize completed milestones for Bewilder, as well as the estimated time to complete each milestone:

1. **Add a way to import and render web links and retrieve web link metadata**: Get potentially nested URLs from any dragged-in `.webloc`, `.url`, or HTML file, as a JSON object within JavaScript. Make a "dropzone" component to detect file drag and drop within the browser window. For each web item, initialize a node-like JavaScript object, and populate its attributes with a UUID, retrieved and scraped Open Graph Protocol metadata, an initial `level`, initial rendering positions, or otherwise. Render a trivial HTML element for each node. *Began October 19th, revised October 30th, and completed November 10th. Some revisions to uniquely identify nodes with UUIDs, add support for multi-file imports, add adding support for downloading the JavaScript data object as JSON, were made in December. Approximately 28 hours.*

2. **Create radial and graph views**: Render a radial force-directed graph layout with a partially ordered hierarchy, and set node radii based on whether they are leaves or not, which is described in more detail above. Set node colours. Add drag-and-drop interactivity to place nodes within different annuli. Add a way to enter the annotation view. Fine-tune force-directed graph parameters to prevent rendering "blowouts" or distracting motions. Add colour schemes and link marks to radial and graph views. *Began November 1st, concurrently with the third milestone, and completed December 8th. Later contributions and revisions centred on bug fixes, for example the graph state being duplicated, and also making "source" nodes larger when they have dependent nodes, adding link directionality based on sources, tuning link length to cluster nodes, adding undirected connections for categories, and making default links have dashed lines. Approximately 66 hours.*

3. **Add features: annotate, arrange, connect, etc.**: Add a node hovering text component or tooltip feature, especially for the `title` attribute of each node. Show annotation view when a node is held for a moment. For each node, add a way to edit categorical and directed node and link attributes, respectively, through the annotation view. Update nodes and links when categories or sources change. Add "default connections" control. *Began November 1st, concurrently with the second milestone, and completed December 8th. Later contributions and revisions centred on bug fixes, for example re-render issues when switching views, and also replacing a tree-like view for neighbouring "sources" and "targets" with a tooltip-like text component, adding support for user-editable* `notes` *and* `categories` *attributes, adding "default connections" for similar base URLs, adding a* `sources` *list to the annotation view, and making changes in the annotation view automatically re-render the graph view. Approximately 24 hours.*

4. **Prepare presentation, code, and report**: (as described). *Began December 6th, and completed December 14th. Approximately 24 hours for the video, 4 hours of code refactoring, and 36 hours for the report.*

5.3 Results

To demonstrate Bewilder, we can return to needs of User (2): to annotate, re-arrange, and add notes to many interdependent web items. With a few modifications this use case scenario also describes the needs of Users (1), (3), and (4).

We suppose that this student or researcher has a collection of web links, either exported as an HTML bookmark file, or stored as tabs saved as multiple `.webloc` or `.url` files. The user drags one of these files directly into the browser window, and a message appears to indicate that the browser recognizes a file dragging event. This message is shown in Fig. 10. As mentioned, dragging in a file produces a JavaScript data object containing nodes, with metadata as attributes, including scraped Open Graph Protocol metadata, and links, including ones derived as "default connections," which are rendered as a force-directed graph.

The user can hover over any node to see a text display of the titles of nodes that are connected to it. As mentioned, on the left are "source" nodes, and on the right are "target" nodes. The `title` of the hovered-over node and a subset of its editable metadata, in particular `categories` and `notes`, are displayed below the radial view.

Since the node hierarchy is not strictly enforced, the user can freely change the `level` attribute of each node by simply dragging and dropping it into a different annulus. Doing so updates the underlying JavaScript data object accordingly. The user may, for example, place a web item that marks the beginning of some research direction in the centre of the radial view to emphasize its centrality, or to indicate that it is the "origin" for something. Other web items can then be re-arranged and placed in reference to this central node. Of course, the user may find it more evocative to indicate the *end* of some research direction with the centre, and precursors to this endpoint in higher and higher hierarchical levels. This interpretation is feasible too.

To annotate a web item, the user clicks and holds a node for a moment, which reveals the annotation view, shown in Fig. 9. The

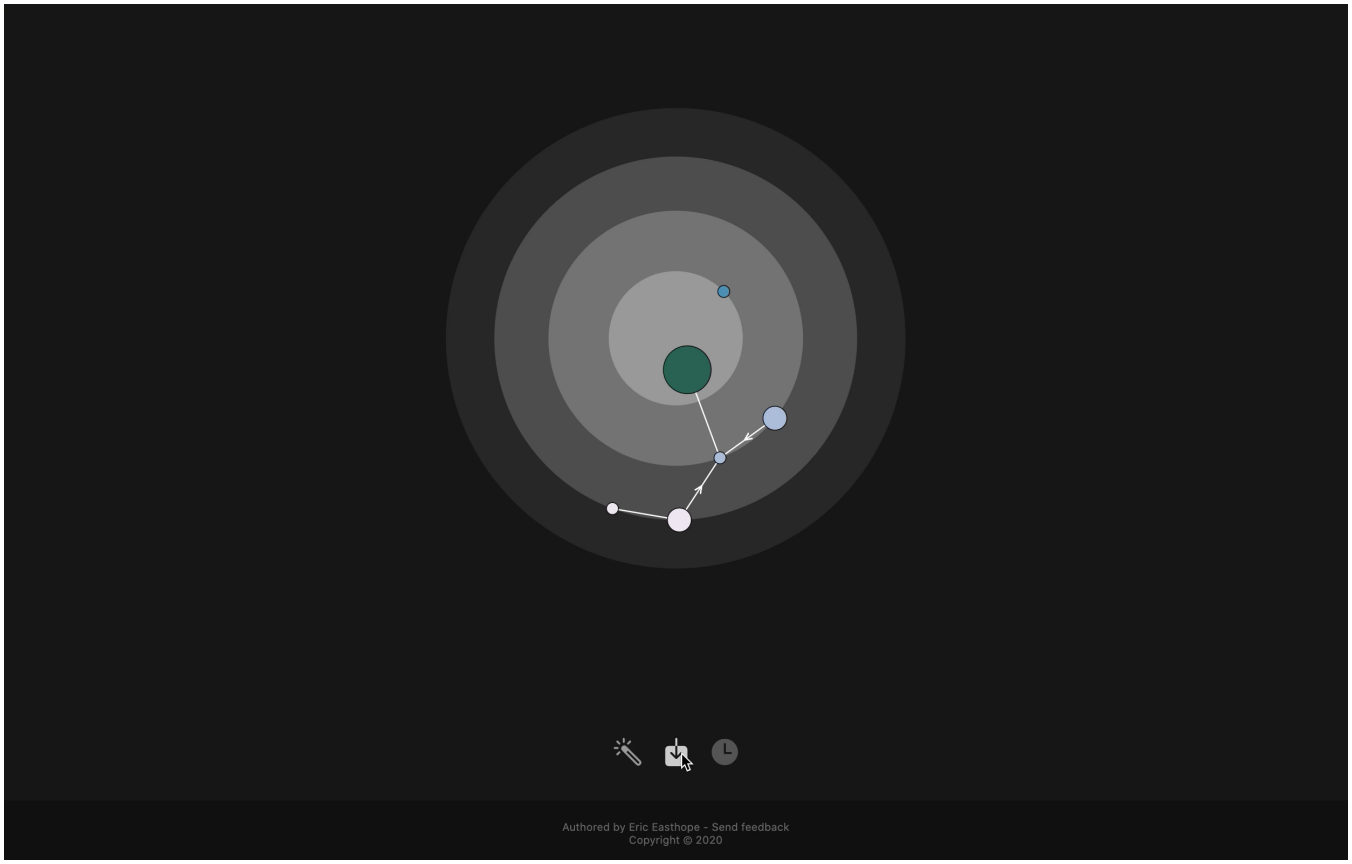

Figure 8: Radial hierarchy and force-directed graph view in Bewilder, showing “default connections,” undirected links derived from node `categories` attributes, and directed links derived from node `sources` attributes. Controls for toggling “defaults connections,” downloading the JavaScript data object as JSON, and toggling “chronology” (disabled), are also shown.

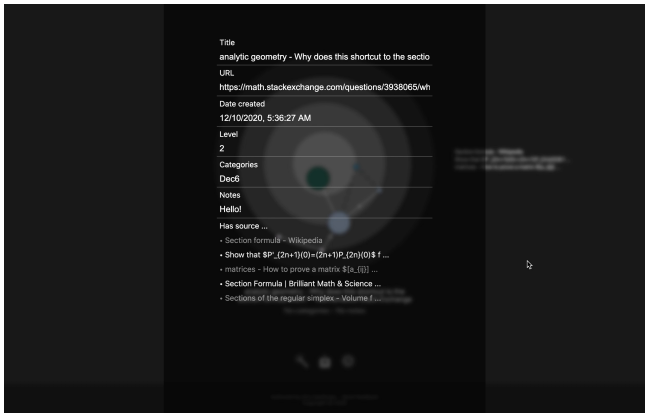

Figure 9: Annotation view in Bewilder.

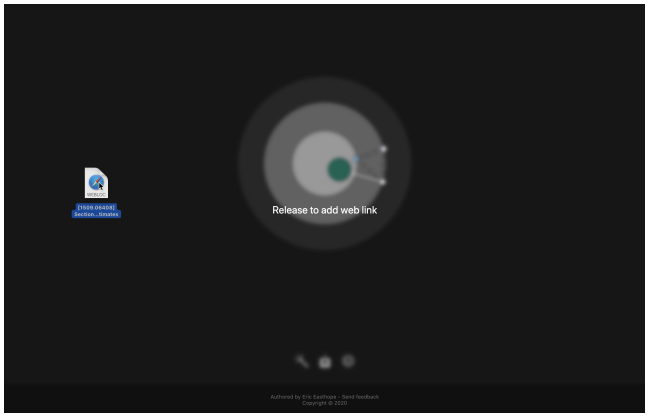

Figure 10: Bewilder recognizing a file dragging event.

`level` attribute of this node can be manually changed here, and brief notes can be added. Also, a list of categories, with categories separated by commas, can be modified, and “sources”, a list of the other web items, can be toggled on or off to produce directed links between nodes. This way, links point from “sources” to “targets.” Toggling other nodes as sources also updates the `sources` attribute of the node being annotated. These links are derived automatically as in Fig. 8 when categories are matched between nodes, or their sources change.

The user can then reiterate this process to add further and further complexity. By toggling “default connections,” the user can also identify web items without annotations as disconnected outliers. One of these outliers is visible in Fig. 8.

If the user wishes to save their changes, they can also download the underlying JavaScript data object as a JSON file, which is shareable and software portable.

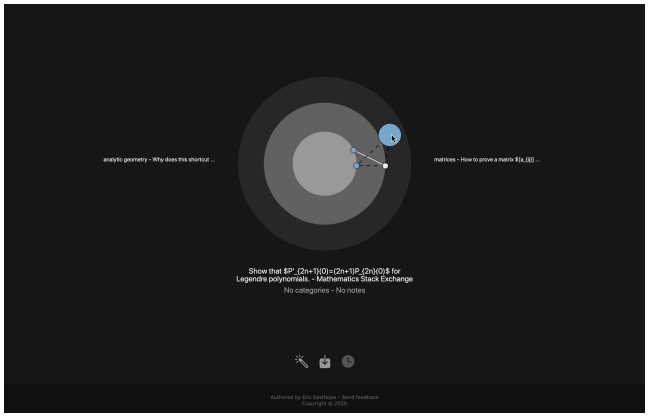

Figure 11: Hovering over or dragging a node reveals its title, its "source" titles, its "target" titles, its `categories`, and its `notes` attribute.

6 Discussion & Future Work

Having a tool that can retrieve, modify, and export both web item metadata and derived connections between web items means that another user can receive, overview, and even add their own content to a readily shareable and well-structured web link collection. Bewilder also makes connections between web items explicit, and therefore adds valuable higher-order structure to any web link collection. These connections, as they are able to be undirected, directed, or defaults, emphasize different relationships between node metadata, which are unattainable with conventional in-browser tabs.

There are some features that were considered and prototyped during development, but were ultimately deemed unnecessary or removed altogether:

- We initially considered using a multi-coloured matrix view as the primary graphic, with node hovering revealing a "local" graph topology composed of the "source" and "target" nodes in Bewilder. However, we decided to emphasize the graph topology since many of these datasets start off quite small. We are in control of the number of the links, and therefore can avoid scenarios where the number of links greatly exceeds the number of nodes, which would motivate a matrix-like view. Matrix views also seemed less readable to new users. Moreover, an investigation by Ghoniem et al. found that node-link views are more effective for some tasks than matrix views if the graphs involved are small [8].

- Gradients of link opacity were tried in place of arrows to define link directions. This did reveal a number of bugs in how we generated links for "default connections," namely that "sources" got mixed up with "targets." The gradients were evocative and had a pleasant aesthetic, but were also computationally intensive to render.

- We tried adding a radial force to set node positions instead of fixing node positions within annuli. This did not work well and created a lot of unwanted motion and graph instability. Though, adding a radial force at the edge of the window seemed to prevent some node overlap. Still, we think that fixing node positions within annuli more effectively communicates a partially ordered hierarchy.

- Link tooltips and annotation, such as for link directionality, were considered. We determined that in denser graphs it would be too difficult to accurately interact with specific links.

- We considered de-emphasizing disconnected nodes by decreasing their opacity, to signify dissimilarity in their metadata to other nodes. However, we decided that it was not meaningful enough to distinguish outliers in this way.

- We prototyped a feature where hovering over an annulus would reveal the titles of all nodes with the same `level` attribute. This also did not seem meaningful enough since the `level` attributes are justified by the user and are not intrinsic to specific web links.

There are also a number of visualizations features that remain incomplete or we think deserve attention in future work. Some of these intended features, as they fall under UI development, are scoped outside of this paper. These UI-based features are italicized here:

- The "default connections" feature should derive connections from retrieved Open Graph Protocol metadata, instead of just comparisons of base URLs. Since this metadata varies greatly between webpages, it is more challenging to do this than simply making keyword comparisons. Also, creating connections based on web links sharing base URLs can sometimes result in unmanageably complex graphs.

- Given our task descriptions, Bewilder should produce overviews for as many as hundreds or more annotated and interlinked web items. This means there is more work to be done to ensure that graphs with many more links than nodes do not become unreadable nor unusable, as seen in Fig. 12.

- In the case of more complex graphs, more force constraints should prevent unwanted link crossings. Aggregating links could also be used to de-emphasize link quantity but still emphasize connections between nodes. Reducing graph sizes by aggregating nodes with sufficiently similar metadata could also improve graph readability and ease of dragging multiple nodes between hierarchical levels.

- To satisfy Users (3) and (4), a "chronology" feature should at least partially enforce node `level` attributes based on when their corresponding web links are dragged into Bewilder. This would require some partitioning of timestamps into a finite number of possible `level` attribute values, where the first web link added is either placed at the centre or furthermost annulus, and the most recently added web link is placed opposite to the first.

- *Users should be able to load a previously exported JSON file with Bewilder, in order to continue working on a saved graph.*

- *It should be easier or at least more obvious how to exit the annotation view, and users should be given multiple ways to do so. For example, a user could press a "$\times$" symbol in one of the view corners.*

- *The annotation view should better emphasize which metadata, such as the* `categories` *attribute, is being actively used to generate links.*

One downside of exporting the entire JavaScript data object as JSON is that this JSON file requires more file space than many HTML, `.webloc`, or `.url` files. However, we think this is less of a pressing issue with increasing file storage capabilities on most personal and workplace computers.

Finally, a last-minute 1-person user study with Bewilder, which was accessed through Google Chrome, revealed an unexpected use case. After watching a silent demo video, the user had saved an

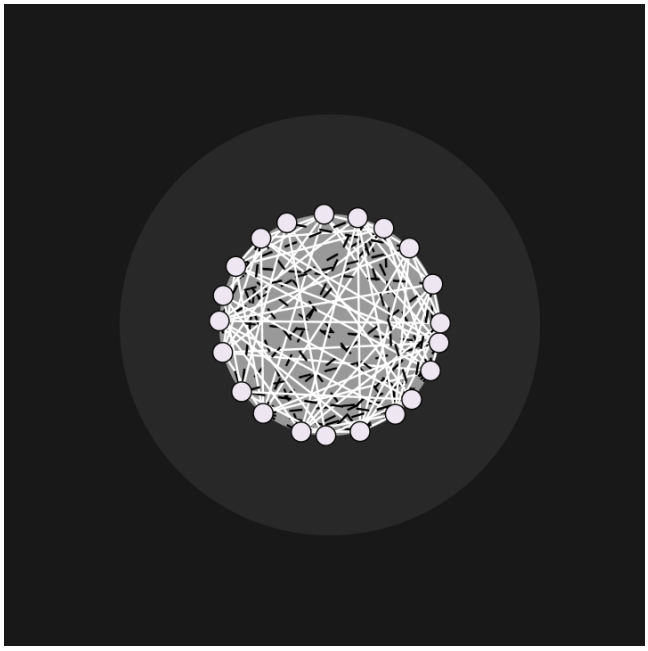

Figure 12: Reduced readability when there are too many "defaults connections" and undirected `categories` links.

entire webpage as an HTML file, and dragged this file into Bewilder. Bewilder, albeit not crashing, rendered one node for the webpage, as well as for every embedded HTML element within it. This included HTML for ads, all linked by similarities in base URL. The user also found another webpage that did not consist of HTML and therefore Bewilder could not retrieve metadata for it. Most importantly, without guidance the user did not initially interpret the annuli as a hierarchy. As the use of this hierarchy is primarily justified by the user, this confusion on their part is expected. Entering the annotation view by clicking and holding a node for a moment was also not self-evident from the video demo alone.

Nevertheless, with more guidance, the user was able to understand use cases for the tool, especially when it was described as a means to "... visualize a thought process behind collecting resources." The user also said that Bewilder seemed useful for "tracing resource paths followed to get to an answer or [end resource]." Unexpected use cases like the one mentioned make us look forward to further user exploration and feedback.

7 Conclusion

We have presented Bewilder, an online tool for annotating, rearranging, and connecting web links in a partially hierarchical way based on intrinsic and retrieved web link metadata represented by a force-directed graph. This tool derives from previous insights into radial hierarchical views and small graphs, as well as personal frustrations with existing software tools.

Bewilder makes node metadata editable through an annotation view, and derives undirected and directed connections from similarities in node metadata. Connections are emphasized through various default or user-specified undirected and directed link marks, and identified through user click, hold, and drag interactions with nodes. Users can also download modified web link metadata as a JSON file. More work remains to be done to fulfill some user story requirements and the UI development directions described.

Acknowledgments

I am deeply grateful to Professor James Colliander for his patience, mentorship, and for giving me the time and space to learn D3.js when I was an undergraduate at UBC.

References

[1] D. Archambault, T. Munzner, and D. Auber. GrouseFlocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):900–913, 2008. doi: 10.1109/TVCG.2008.34

[2] S. Chaudoin, J. N. Shapiro, and D. Tingley. Revolutionizing teaching and research with a structured debate platform. Working paper, 2017. Accessed: 2020-12-14.

[3] C. Collins, S. Carpendale, and G. Penn. DocuBurst: Visualizing document content using language structure. *In Proc. Eurographics/IEEE-VGTC Symp. Visualization (EuroVis)*, 28(3):1039–1046, 2009. doi: 10.1111/j.1467-8659.2009.01439.x

[4] S. Fabbri, E. Hernandes, A. Di Thommazo, A. Belgamo, A. Zamboni, and C. Silva. Managing literature reviews information through visualization. *In Proc. Intl. Conf. Enterprise Information Systems (ICEIS)*, 2:36–45, 2012.

[5] Facebook, Inc. Open Graph Protocol. `https://ogp.me`, 2010. Accessed: 2020-12-14.

[6] S. Fast. Introduction to the Zettelkasten method. Blog, 2020. Accessed: 2020-12-14.

[7] C. Friedrich and F. Schreiber. Flexible layering in hierarchical drawings with nodes of arbitrary size. *In Proc. Australasian Computer Science Conf. (ACSC)*, pp. 369–376, 2004.

[8] M. Ghoniem, J. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. *In Proc. IEEE Symp. Information Visualization (InfoVis)*, pp. 17–24, 2004. doi: 10.1109/INFVIS.2004.1

[9] S. Hong and N. Nikolov. Hierarchical layouts of directed graphs in three dimensions. *Lecture Notes in Computer Science*, 3843:251–261, 2006. doi: 10.1007/11618058_23

[10] G. King. An introduction to the Dataverse Network as an infrastructure for data sharing. *Sociological Methods and Research*, 36:173–199, 2007.

[11] Mozilla Corporation. Pocket. `https://getpocket.com`, 2007. Accessed: 2020-12-14.

[12] R. Mussabekov. Raindrop.io. `https://raindrop.io`, 2013. Accessed: 2020-12-14.

[13] S. C. North and G. Woodhull. Online hierarchical graph drawing. *Revised Papers from Intl. Symp. on Graph Drawing (GD)*, pp. 232–246, 2001. doi: 10.1007/3-540-45848-4_19

[14] Oak Studios. Dropmark. `https://www.dropmark.com`, 2011. Accessed: 2020-12-14.

[15] Obsidian. Obsidian. `https://obsidian.md`, 2020. Accessed: 2020-12-14.

[16] Roam Research. Roam. `https://roamresearch.com`, 2019. Accessed: 2020-12-14.

[17] Roy Rosenzweig Center for History and New Media. Zotero. `https://www.zotero.org`, 2006. Accessed: 2020-12-14.

[18] A. Saghafi. Visualizing ontologies – a literature survey. *In Proc. Intl. Conf. Conceptual Structures (ICCS)*, pp. 204–221, 2016. doi: 10.1007/978-3-319-40985-6_16

[19] A. Singhal. Introducing the Knowledge Graph: things, not strings. Blog, 2012. Accessed: 2020-12-14.

[20] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald. An evaluation of space-filling information visualizations for depicting hierarchical structures. *Intl. Journal of Human-Computer Studies (IJHCS)*, 53:663–694, 2000. doi: 10.1006/ijhc.2000.0420

[21] C. Tietze and S. Fast. The Archive. `https://zettelkasten.de/the-archive`, 2017. Accessed: 2020-12-14.

[22] S. N. Tumpa and M. Masroor Ali. Document concept hierarchy generation by extracting semantic tree using knowledge graph. *In Proc. IEEE Intl. WIE Conf. Electrical and Computer Engineering (WIECON-ECE)*, pp. 83–86, 2018. doi: 10.1109/WIECON-ECE.2018.8783083

[23] UnaMesa Association. TiddlyWiki. https://tiddlywiki.com, 2004. Accessed: 2020-12-14.