ARTICLE

Knowledge Graph Visualization Interface for Digital Heritage Collections

Design Issues and Recommendations

Christopher S. G. Khoo, Eleanor A. L. Tan, Siam-Gek Ng, Chwee-Fong Chan, Michael Stanley-Baker, and Wei-Ning Cheng

ABSTRACT

Digital heritage portal interfaces are generally similar to digital library and search engine interfaces in displaying search results as a list of brief metadata records. The knowledge organization and search result display of these systems are item-centric, with little support for identifying relationships between items. This paper proposes a knowledge graph system and visualization interface as a promising solution for digital heritage systems to support users in browsing related items, understanding the relationships between items, and synthesizing a narrative on an issue. The paper discusses design issues for the knowledge graph, graph database, and graph visualization, and offers recommendations based on the authors' experience in developing three knowledge graph systems for archive and digital humanities resources: the Zubir Said personal archive collection at the Nanyang Academy of Fine Arts, Singapore; Singapore Pioneers social network; and Polyglot Medicine knowledge graph of Asian traditional and herbal medicine. Lessons learned from a small user study are incorporated in the discussion.

INTRODUCTION

Digital heritage portal interfaces are generally similar to digital library, bibliographic retrieval system and search engine interfaces in displaying search results as a list of brief metadata items, with each item hyperlinked to a more detailed metadata record display which may include the full text or image of an information resource. Thus, digital heritage and digital library systems are item-centric in design: users are expected to examine retrieved records one at a time, either in a list or on separate web pages. The knowledge organization scheme, the database structure and the interface of these systems are not designed to help users identify conceptually related resources and ideas, except for a collection field indicating association with a particular context (e.g., institution, person, event, etc.), or a subject field with keywords, subject headings (from a controlled vocabulary), or class code (from a classification scheme). These systems do not support browsing by specific types of relationships such as different arrangements or performances (expressions) of the same work, familial and business relationships between persons, and a chain

About the Authors

Christopher S.G. Khoo (corresponding author: chriskhoo@pmail.ntu.edu.sg) is Associate Professor, Nanyang Technological University, Singapore. **Eleanor A.L. Tan** (altan@nafa.edu.sg) is Principal Lecturer, University of the Arts Singapore. **Siam-Gek Ng** (ng0001ek@e.ntu.edu.sg) is LIBRES Editorial Assistant, Nanyang Technological University, Singapore. **Chwee-Fong Chan** (cs-chweefong.chan@ntu.edu.sg) is LIBRES Editorial Assistant, Nanyang Technological University, Singapore. **Michael Stanley-Baker** (msb@ntu.edu.sg) is Assistant Professor, Nanyang Technological University, Singapore. **Wei-Ning Cheng** (ivanka@ntnu.edu.tw) is Assistant Professor, National Taiwan Normal University, Taiwan.

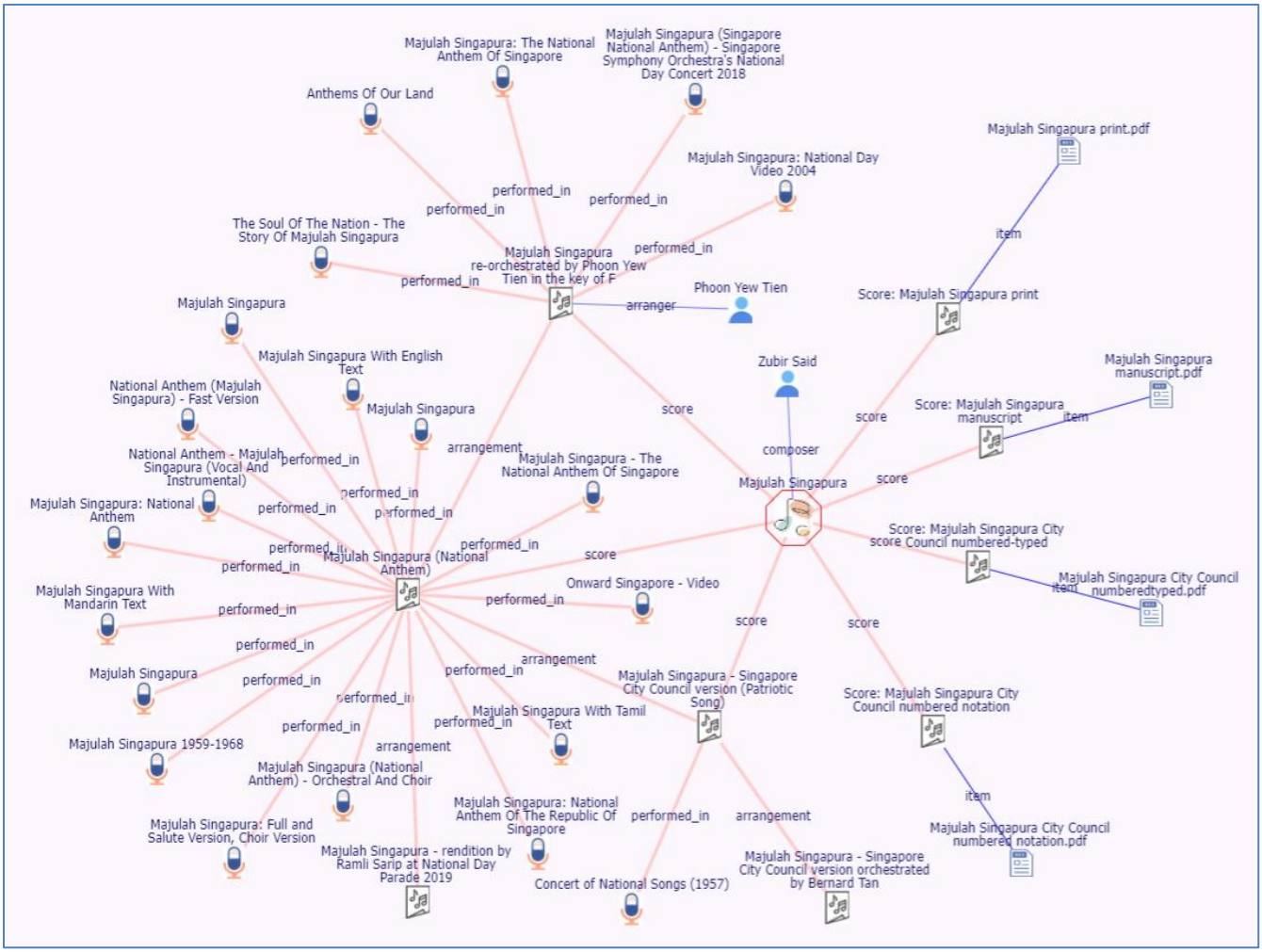
This peer-reviewed work was submitted 15 July 2023, accepted for publication 13 November 2023, and published 18 March 2024.

https://doi.org/10.5860/ital.v43i1.16719

of correspondence; nor can they identify sets of items with particular relationship patterns to help users synthesize a coherent narrative from the set of related records.

This paper proposes a knowledge graph system and visualization interface as a good solution for digital heritage systems, and more generally digital humanities collections, to support users in associative browsing (i.e., browsing related entities and concepts), in linking and integrating information, and in synthesizing an overall understanding. The paper discusses design issues for such systems, focusing on the graph visualization interface, and offers design recommendations based on our experience in developing three knowledge graph systems. The paper is written to guide digital archive and digital humanities researchers and digital librarians in developing knowledge graph visualization interfaces.

Figure 1. Graph visualization of resources centered on Majulah Singapura, the Singapore national anthem.



We use as the main case study the Zubir Said personal archive at the Nanyang Academy of Fine Arts (NAFA), Singapore. Zubir Said (1907–1987) was the composer of the Singapore national anthem and a pioneer musician in Singapore after World War II. His family donated his personal archive to NAFA, and the authors of this paper were invited to develop a web-based system to provide access to digital resources from the collection to support study and research. Consequently, we developed the Zubir Said knowledge graph system (https://ZubirSaid.sg).

Figure 1 shows a graph visualization of resources centered on Majulah Singapura, the Singapore national anthem. In addition to the Zubir Said knowledge graph system, we also draw insights and lessons from developing two other knowledge graph systems in digital humanities domains:

- Singapore Pioneers knowledge graph and social network (https://SingPioneers.sg): for visualizing social networks of famous persons in post-war Singapore, derived from the biography pages of the Singapore Infopedia (the Singapore National Library's electronic encyclopedia on Singapore); and
- Polyglot Medicine knowledge graph (https://kgraph.sg/polyglot): a knowledge graph of Asian traditional and herbal medicines, linked to literary sources, historical, geographic, and scientific information, and external database records.

THE KNOWLEDGE GRAPH SYSTEM

Though this paper focuses on the design of the web interface and graph visualization, the content and design of the underlying knowledge graph, the capabilities of the graph database management system, and the software packages used have a substantial impact on the interface and graph visualization design. This section provides an overview of the knowledge graph system, focusing on the aspects that have important implications for the interface display.

The three knowledge graph systems used as case studies have a 3-layer architecture (see fig. 2):

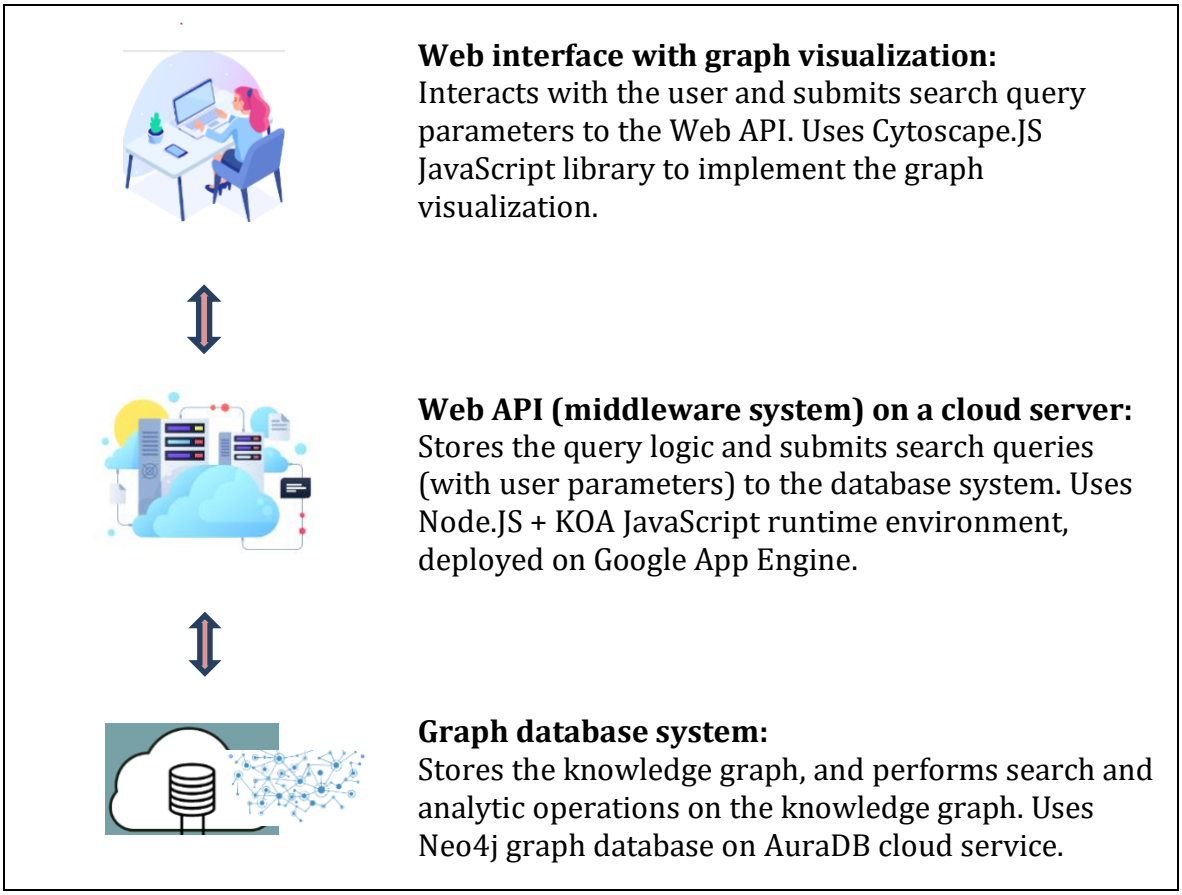
1. a graph database management system storing the knowledge graph and performing search and analytic operations on the knowledge graph;
2. a web API (Application Programming Interface) serving as a middleware system, performing additional data processing and mediating the interaction between the graph database system and the web interface; and
3. a web interface with a graph (network) visualization function.

The main technologies used for each layer are listed in figure 2. This section provides an overview of some of these technologies and outlines how their design and implementation affect the interface display and graph visualization.

The information content and structure (or topology) of the graph visualization closely follow the knowledge graph (i.e., the knowledge organization) that is constructed to serve as a metadata layer to describe resources in the digital collection, the relations between them, and relations to important concepts as well as to web resources outside the collection. There is no consensus or standard definition of the term *knowledge graph*. In this section, we provide our perspective and working definition of knowledge graph.

As the name implies, a knowledge graph employs a graph (i.e., network) structure of nodes (or vertices) and directed links (also called *edges* or *arcs*, often displayed as arrows) to represent an information/knowledge structure. The graph structure is assigned semantics or meaning by assigning concept or entity labels to nodes and relationship types to the links. Some kind of knowledge structure is often imposed on the graph by linking concept and entity labels to taxonomies or class hierarchies (which are also represented in graph format). The term knowledge graph is thus similar to ontology, semantic network, semantic web, linked data, and Resource Description Framework (RDF), though each term emphasizes different aspects of knowledge representation, different applications, and different research communities.

Figure 2. Knowledge graph system architecture. (Image credit: iStock.com. Use under license.)



The term knowledge graph has been traced back to Schneider, who suggested that the body of knowledge that the instructor seeks to impart to students can be represented in simplified form as a knowledge graph.[1] Google popularized the term in 2012 by constructing a network of concepts and entities derived from Wikipedia, DBpedia, and other sources as a supplement to keyword searching.[2] Bergman provided an extensive survey of knowledge graph definitions, and Hogan et al. have offered a historical overview of the concept and term.[3] Ehrlinger and Wöß reviewed extant definitions of the term knowledge graph and proposed that "a knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge."[4] This capability of a knowledge graph to support inferencing to derive new knowledge is important for intelligent applications.

However, we propose an alternative definition of knowledge graph that distinguishes it from ontology and emphasizes support for human information seeking and information use. We propose that the focus of a knowledge graph is less on logical reasoning, but more on linking things in a graph (network) representation. We also associate a knowledge graph with social network analysis. Logical reasoning to infer new information (i.e., new nodes, links, or properties) can be performed on a knowledge graph, but this is based less on logic and more on graph pattern matching and social network analysis using the query language of the graph database management system. The growth of graph databases has stimulated interest in these aspects of knowledge graphs. We informally characterize a knowledge graph as a "social" network of resources (ideas/concepts and entities) and the semantic relations between them, represented as

a network of nodes connected by directed links. The nodes are assigned meaning by labeling them with classes in a taxonomy and assigning them properties (attributes). The links are also labeled with relationship types and may be assigned properties as well.

As there is no international standard or industry specification for knowledge graphs, the features of a knowledge graph and the operations that can be performed on it depend on the graph database management software used. Graph database software can be divided into two main types, following two different knowledge graph models: those based on triple-stores such as Resource Description Framework or RDF and OWL2 Web Ontology Language, and those based on the labeled property graph model. Barrasa and Feeney have discussed the differences.[5] Our knowledge graphs are implemented on the Neo4j graph database management system based on the labeled property graph model. As this model is less well-known than RDF/OWL2, we summarize the main differences below.

A major difference is that labeled property graphs can store properties in the links (relations). In RDF and OWL2, relations cannot be assigned properties. To add a property to a relation, the relation must be represented as an intermediate node, linked (by more primitive relations) to the source and target nodes. This intermediate node will also allow it to have more than two relations to other resources (i.e., an n-ary relation).[6] Representing relations as intermediate nodes makes the graph more complex and difficult to understand than representing them with just arrows.

Secondly, there is no built-in distinction between entity (instance) nodes and class nodes in labeled property graphs. So, the developer must carefully distinguish between them in the graph database design.

A third major difference is that a labeled property graph as implemented in Neo4j database software is schema-free (or schema-less) in that node types are not specified with a set of mandatory properties. Domain-range restrictions, property restrictions, and cardinality restrictions used in OWL2 cannot be specified in a labeled property graph.[7] So, any attribute-value pair (property) can be added to any link in a labeled property graph, which makes it flexible but also chaotic—as it is difficult to anticipate what attributes will be in a node or link during system development. Thus, labeled property graphs are simpler and more lightweight than RDF/OWL2 graphs.

The Neo4j graph database management system is one of the most popular and highly-rated graph database software.[8] It is a pioneer in graph database technology with version 1 released in 2010. Min provided a brief but informative history of graph database technology.[9] Newer graph database products (such as Ultipa Graph Database, TigerGraph, and JanusGraph) may be faster, more scalable, and better able to handle transaction data (data generated by business and sales operations) and real-time data. They also employ database schema. However, digital archive and digital humanities collections probably do not require such huge data storage and fast processing, nor transactional or real-time data analytics. In our opinion, Neo4j database amply meets archival and digital humanities data needs. We especially like Neo4j's graph query language (called Cypher), which is more intuitive than SPARQL.

Our graph visualization interface runs in the user's web browser and is the visible part of the system to end users. Graph visualization is a natural way to display a knowledge graph or parts of it: it renders the abstract network structure in a two-dimensional layout called graph visualization. The nodes are assigned properties, including a unique identifier (ID), a label

attribute, and a type attribute. Usually, only the node label is displayed in the graph visualization. The other node properties may be displayed in a separate text display panel (that we refer to as info box).

The design of the graph visualization depends largely on the capabilities and scripting language of the graph visualization software. We adopted the Cytoscape.js JavaScript graph visualization library (https://js.cytoscape.org/) to realize the graph visualization on users' web browsers. Cytoscape was initially developed for visualizing gene expression data.[10] The JavaScript library version, Cytoscape.js, was developed separately for data visualization in web applications.[11]

Other free JavaScript libraries (plugins) for graph visualization include Sigma.js and Vis.js. A few lists and comparisons of graph visualization JavaScript libraries are available on the web.[12] However, it is difficult to compare the effectiveness of visualization libraries without actual implementations. We carried out a small comparison study with a team of graduate students by implementing our visualization interface using multiple visualization libraries. Cytoscape.js was clearly superior to the other visualization libraries we explored. We did not adopt D3.js (a well-known and powerful visualization JavaScript library) or ReactFlow as they require more low-level programming and more coding effort to realize graph visualizations of comparable aesthetics and functionality.

We can recommend Cytoscape.js for implementing graph visualization interfaces for digital heritage and digital humanities system interfaces, and knowledge graph systems in general. The basic graph visualization design is specified as attribute-value pairs (i.e., parameters) in a Cytoscape stylesheet in JSON format. A comprehensive set of JavaScript functions are available to add, remove, retrieve, filter, position, count, and in general, manage the nodes and links, as well as respond to mouse/tap events on graph elements.

KNOWLEDGE GRAPH VISUALIZATION INTERFACES—OVERVIEW & SURVEY

This section discusses the main characteristics of knowledge graph visualization interfaces, and how they are different from other kinds of graph visualizations. Existing knowledge graph visualization interfaces are reviewed.

Most of the graph visualizations seen on the web are standalone visualizations where the whole dataset is loaded onto the user's web browser memory space. In contrast, a knowledge graph system stores the dataset in a graph database management system, and only a relatively small subset of the graph (retrieved by a search query) is displayed or visualized on the interface. Thus, the interface needs to offer user search functions to retrieve a subset of nodes and links of appropriate size to display. But how big is too big to display as a readable graph visualization?

In a survey of graph visualization techniques, von Landesberger et al. found no standard definition of small versus large graphs except that large graphs were expected to look cluttered, take a long time to plot the visualization, and may exceed RAM memory.[13] Yoghourdjian et al. carried out a comprehensive survey of 152 user studies and evaluations of graph visualizations.[14] They pointed out that what is considered "large" or "complex" depends on a few factors including data complexity, visual complexity, and the technology used. Huang et al. pointed out that other factors affect the cognitive load of graph visualizations, including domain complexity, task complexity, demographic complexity, and time pressure.[15]

Based on the graph sizes (number of nodes) used in the user studies reviewed, Yoghourdjian et al. proposed the following size categories of graph visualizations:[16]

1. Small: 20 nodes or less (used by 41% of the 152 studies reviewed)
2. Medium: 21 to 50 nodes (33% of the studies)
3. Large: 51 to 200 nodes (37%)
4. Very large: more than 200 nodes (22%)

The percentages do not add up to 100% because some studies made use of multiple-sized graphs. While 80% of the studies used graphs of 100 nodes or fewer, 74% used graphs with 200 or fewer nodes. Of the studies that made use of very large graphs of more than 200 nodes, 70% offered interaction and aggregation functions to display a smaller graph for user examination.

The link density of a graph is an important factor for its readability. Graph density is the number of links in the graph, divided by the maximum possible number of links between every pair of nodes.[17] In Yoghourdjian et al.'s survey, all the studies that made use of graphs with more than 50 nodes used sparse graphs with a link density of less than 10%.[18]

The above graph size classification by Yoghourdjian et al. is generally in line with our own experience. Our knowledge graph visualizations have the following characteristics that have implications for the readability of different graph sizes:

1. All node labels and link labels are displayed on the graph visualization. The node labels are relatively long and can run to three lines, with a maximum width of 180 pixels.
2. The graph visualization is not bounded by the display canvas (or viewport): the graph display can "burst the seams" and extend beyond the display canvas, which will require the user to pan the display canvas (by dragging the mouse anywhere in the canvas) to bring an off screen part of the graph into view.
3. The interface is assumed to be displayed on laptop or notebook computers or tablets (iPad or Android tablets). Mobile phone screens are generally too small for graph visualizations.

With these caveats, we have found that graphs of 50 nodes or less are comfortable for users to understand and accomplish information tasks. Graph sizes between 51 to 100 nodes are manageable, but it is desirable to offer a filter menu for the user to reduce the number of nodes or links. Graphs of 101 to 150 nodes are manageable for experienced users, who know how to interact with the graph to extract desired information or to visually analyze the graph. We would consider graphs of more than 150 nodes to be unmanageable.

The graphs displayed in our knowledge graph visualizations are very sparse, with a typical link density of 3% (for undirected graphs). A better sense of how dense a graph is can be gained by calculating the average number of links per node, sometimes referred to as linear link density. The linear link density of our graphs range between 1 and 1.5 (i.e., about an equal number of nodes and links).

Graphs can also be categorized into static graphs and dynamic graphs that have a temporal dimension, sometimes called time-dependent or time-varying graphs.[19] The focus of this paper is on static graphs.

As mentioned earlier, digital heritage system interfaces are generally similar to digital library interfaces in displaying search results as a list of metadata records, possibly with an

accompanying illustrative image (e.g., *Europeana*, *Hungarian National Digital Archive*, *Auckland Museum*, and *National Archives of Singapore*). There are a few knowledge graph visualization interfaces that have been implemented for digital heritage and digital humanities content. Most of the graph visualizations are simple and plain, without much effort put into their design and implementation. The focus of these projects was often on the ontology design and the automatic extraction of information from text to populate the ontology.

The Digital Humanities Platform of Shanghai Library has a knowledge graph interface to visualize relations between historical figures in modern Chinese history (see fig. 3). The graph presents a person's relationship with other people, the person's works, historical events he or she was involved in, and the events' locations. Information on each historical figure is assigned to the person node representing the figure and displayed in a text display panel when the node is selected. Different colors are used to signal different types of entities. As illustration, figure 3 presents a knowledge graph centered on Yuan Shikai (袁世凱), a famous Chinese army leader and the first president of the Republic of China. The graph shows his relation to other persons (represented as blue nodes), works (yellow nodes), and historical events (red nodes). The right panel gives his biodata (i.e., name, birth and death years, birthplace, gender, and ethnicity). The bottom part of the right panel offers filters for different kinds of relationships: kinship, friendship and others shown as blue buttons; collaborations shown as a yellow button; and relationship with fellow provincials shown as a green button.

The Malaysia Henghua Personalities system stores records of 85 historical figures who were active in the Malaysian Chinese political, business, cultural, and artistic circles. The knowledge graph represents relationships between individuals, organizations, and events.[20] Figure 4 shows a graph centered on Lin Jinshu (a prominent businessman), with links to other persons (yellow nodes) and organizations (blue nodes). When a node is clicked (i.e., selected), information on all the links to other persons and organizations is displayed in the right panel. The left panel offers drop-down menus to select an individual, an organization, a relationship type, and time period (start and end years) as parameters for a search. The graph visualization design is simple, with different colored nodes.

The Universal Type Digital Humanities Research Platform on Chinese Ancient Books presents information based on a special Ming Dynasty collection in the National Central Library of Taiwan. The system provided the full text of each book in its collection, as well as a graph visualization of the relationships between the figures mentioned in the book. However, the system seems to be no longer accessible (as of January 2024).

Hyvönen and Rantala described a knowledge graph constructed from biographical data of 13,000 historical persons in Finland, linked to digital resources in museums, libraries, and archives.[21] They described their effort to help users find "serendipitous semantic relations between resources in [the] knowledge graph." Essentially, graph operations were used to trace a path in the graph between two specified persons. The authors made used of facets to limit the paths that could be traced between the persons to those that reflected interesting relationships. A very simple graph visualization (https://biografiasampo.fi/verkosto) showing links between two to four persons is provided.

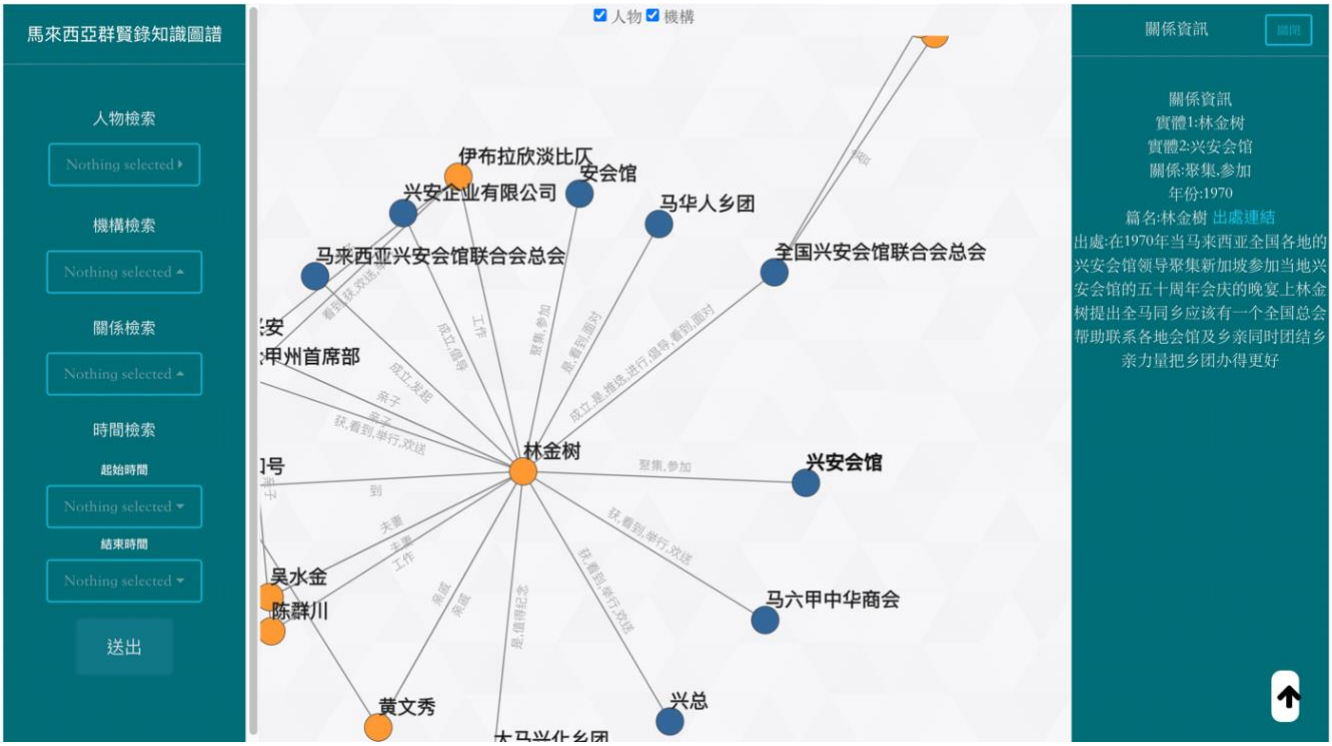
Wu, Jiang, Chen, Guo, Wei, and Yang developed the Canton Revolutionary History Knowledge Graph to represent historical information relating to the Canton Revolution that ended China's last imperial dynasty, the Qing dynasty, leading to the establishment of the Republic of China.[22] The

knowledge graph served as an interface layer above the Canton Canon Database (digital library of historical and archival materials) to guide users to retrieve relevant documents in the digital library. Unfortunately, the system is no longer accessible (as of June 2023).

The Song and Yuan Xuean Knowledge Graph System developed by the Digital Humanities Research Center of Peking University represents relations between Confucian scholars in the Song and Yuan dynasty, and their works, time, and place. The Dunhuang Grotto Knowledge Graph System developed by the Digital Humanities Research Center of Wuhan University represents ontology data related to cultural treasures at the Dunhuang Mogao Grottoes (also known as the Caves of the Thousand Buddhas).[23] The knowledge graph covers data related to the grottoes, murals, sculptures, manuscripts, photographs, models, and research publications.

Figure 3. The graph visualization interface of the Digital Humanities Platform of Shanghai Library.

Figure 4. The graph visualization interface of the Malaysia Henghua Personalities system.



Most of the papers on the construction of knowledge graphs for cultural heritage resources have focused on text mining and entity-relation extraction techniques. Dou, Qin, Jin, and Li described the development of a knowledge graph to integrate intangible cultural heritage resources across many websites in China, using text mining and deep learning approaches.[24] Ellul, Azzopardi, and Abela described the construction of a knowledge graph of historical notarial manuscripts called NotaryPedia, and their efforts in entity-relation extraction from medieval Latin texts.[25]

We did not come across papers that discuss the design of knowledge graph interfaces and graph visualizations to support user browsing, information integration, and analysis of digital heritage and digital humanities resources.

KNOWLEDGE GRAPH, INTERFACE, AND GRAPH VISUALIZATION—DESIGN ISSUES AND RECOMMENDATIONS

This section is the heart of the paper, where we discuss the design of the knowledge graph, the web interface, and the graph visualization. A user study involving 20 participants who were asked to use the Zubir Said knowledge graph to obtain relational information to answer factual questions was completed. A second part of the user study asking the users to outline a narrative on a topic related to the knowledge graph is in progress. A separate paper will report on the user study in detail. However, the design recommendations offered in this section will take into consideration lessons learned from the user study.

Knowledge Graph and Graph Database Design

The graph database is the heart of the system, and the knowledge graph is the information content of the graph database. They determine the structure (topology) and semantic content of the graph visualization. This section presents an overview of the knowledge graph and database design, highlighting design decisions that substantially impact the graph visualization.

In a Neo4j graph database, a node can represent a class or a class instance (i.e., entity), and a directed link represents a relationship instance. Nodes are assigned zero, one, or more *LABELS*, and links are assigned one *TYPE*. We use the uppercase *LABEL* and *TYPE* to distinguish these Neo4j concepts from *rdfs:label* and *rdf:type* in the RDF/OWL2 model. Note that Neo4j node *LABEL* can be used in the same way as *Class* in RDF/OWL2, while link *TYPE* is equivalent to *owl:ObjectProperty*. In addition to the built-in LABELs and TYPEs specified for nodes and links respectively, we have found it useful to add a separate *type* property in all nodes and links. This allows us to enter the most specific class or relation in the *type* attribute, and the top-level class or relation in the built-in node LABEL and link TYPE. Here is an example concept-relation-concept triple specified in Neo4j Cypher language:

```
(:PERSON {id: 'ZubirSaid', label: 'Zubir Said', type: 'Composer'})
        -[:CREATE {type: 'compose'}]->
   [:CREATIVE_WORK {id: 'MajulahSingapura', label: 'Majulah Singapura',
   type: 'National_Anthem'}]
```

This statement, in effect, "draws" a graph path using a text language. Text in parenthesis indicates a node and its properties, and text in square brackets with an arrow indicates a link and associated properties. The Cypher statement specifies that *PERSON:ZubirSaid CREATE CREATIVE_WORK:MajulahSingapura*. *PERSON* is the LABEL (i.e., Class) for the first node. We also add a *type* attribute to the PERSON node with the more specific *Composer* class. Similarly, the link TYPE is *CREATE*, to which we add a more specific *type* attribute with the value *compose*. As the Neo4j database does not have a built-in inference engine to perform inferencing with the *SubClassOf* relation, storing both the top class and the most specific class in the node gives us the option to specify either the most specific class/relation or the top-level class/relation in graph queries and pattern matching operations. The example triple also highlights that all the nodes in our knowledge graphs have three mandatory attributes *id, label*, and *type*, and a mandatory *type* attribute for all links.

We initially modeled the Zubir Said knowledge graph as an ontology in OWL2 Description Logic language, using the TopBraid Composer software. The ontology was then exported to Turtle format, converted into a Microsoft Excel file, and then imported into a Microsoft Access database. The Microsoft Access database was useful in enforcing referential integrity, ensuring that links from a node to other nodes (entities, concepts, classes) are linking already existing nodes. However, the data in a Microsoft Access database cannot be uploaded directly to a Neo4j database. The database tables must be exported to CSV files; and Neo4j Cypher scripts must be written to upload and map the data to the nodes and links in the graph database.

Though Microsoft Access is a powerful desktop database application, we found it difficult to use the software interface for data entry, even with views and data entry forms to support the task. We have since transferred the CSV files to Google Drive, and we now use Google Sheets for data entry, which is the approach we recommend. To check for referential integrity, we use the *VLookup* formula to lookup corresponding values in another Google Sheet. During data upload to the graph database, we also use Neo4j Cypher queries to check that the links refer to existing nodes. We have found that spreadsheets are what digital humanities researchers are comfortable with, both for data modeling and data entry. It is especially convenient for collaborative projects where collaborators from multiple locations contribute data to the project. Google Drive's access control features are useful in controlling different types of access, and the file version history and cell edit history functions make it possible to review edits and recover an earlier version of the spreadsheet. Google Sheets has a publish-to-csv function[26] that dynamically converts a

spreadsheet to a CSV file at a specified URL. This makes it possible to submit a Cypher script to a Neo4j database to retrieve the CSV file from the specified URL for upload to the database.

For a complex dataset with several types of entities and relations, as well as complex relationship patterns, we recommend that the knowledge graph be first designed as an ontology using an ontology editor and then converted to a labeled property graph when the design is stable. This is especially useful if the dataset is not yet collected and the data structure can be designed from scratch, as opposed to a large dataset that is already stored in spreadsheets. As the Zubir Said personal archive contains many types of documents, including music notations of songs, photographs, letters, personal documents (e.g., passport) and audiovisual materials (e.g., tapes, vinyl records), we imported the following existing ontologies into the Zubir Said ontology: Music Ontology (modelled after FRBR), Schema.org, DBpedia, CIDOC-CRM, Biographical Ontology, IPTC Photo Metadata Standard 2019.1, Simple Event Model, and Dublin Core Terms.

It soon became obvious that the ontology was too complex for users to browse, and some information such as namespace URIs will just mystify the user. As the imported ontologies overlap and some classes (e.g., *Person*) occur in multiple ontologies, *type* links from entities to multiple *Person* classes in different imported ontologies will also confuse the user. The complete FRBR structure when applied to music and performance is too complicated for users, as they have to traverse many links to get from the *MusicalWork* to *Expression* (music score), to perhaps another arrangement of the music score, to *Performance*, to *Sound* or *VideoRecording*, to *Manifestation* (perhaps a published DVD), and finally to *Item* to access the *AudioFile*, *VideoFile*, *ImageFile*, PDF file, or webpage. To simplify the structure, we dropped *Manifestation* nodes, *Sound* and *VideoRecording*, and instead link the *Expression* nodes (i.e., music score) directly to the *VideoFile* and *AudioFile* nodes. Initially we stored the hyperlinks to digital files on the *Item* nodes (including the subclasses *VideoFile* and *AudioFile*), but decided to place them also in the *Expression* nodes so that the user can access the digital files without having to traverse or expand more links to get to *Item*.

Node and link labels to display in the graph visualization need to be carefully considered. Node labels are decided when the knowledge graph is constructed. However, link labels are usually relation types possibly imported from another ontology. For example, in the FRBR framework and Music Ontology, the relations *realization* and *manifestation* are mystifying to the casual user. In this project, we display *score* instead of *realization* as the relation usually points to a node of type *Score*.

Finally, too many links in a knowledge graph, especially crisscrossing links, will result in a cluttered visualization that is difficult to read. Although all nodes should have at least one link to another node, links (and the semantic relations they represent) should be added with a clear idea of how they will help the user's understanding, keeping in mind other links in the node neighborhood. We have found it useful to visualize the neighborhood of the linked nodes to check whether the graph is too cluttered.

Web Interface—Overall Structure

The web interface is expected to contain four main parts:

1. A search menu panel—listing canned queries, a keyword search box, and other means for the user to specify a query or node type to retrieve;
2. A graph visualization canvas—to display search results as a graph visualization;

3. One or more text display panels—to display text, including the properties associated with a selected (i.e., clicked) node, as well as search results in text format; and
4. One or more filter menus—to allow the user to specify constraints, to reduce the set of nodes and links in the display.

Figures 5 and 6 show two web interface structures. The ZubirSaid.sg interface (see fig. 5) shows the graph visualization canvas occupying most of the screen, with a pop-up text display box to display attributes of a selected node and a sliding panel on the right offering node/link filter options. A simple menu panel is placed on the left. The Polyglot Medicine interface (see fig. 6) shows a smaller graph visualization panel occupying a less prominent location (but with the option to open fullscreen). A main text panel (above the graph visualization) displays properties of a selected node, but tabbed boxes on the right display linked (related) nodes. The search menu panel on the left is currently partially hidden, but slides open on mouse-over.

Figure 5. ZubirSaid.sg web interface showing the parts of the interface: search menu panel (left), graph visualization canvas (center), pop-up text display panel (draggable), and filter menu panel (sliding from the right).

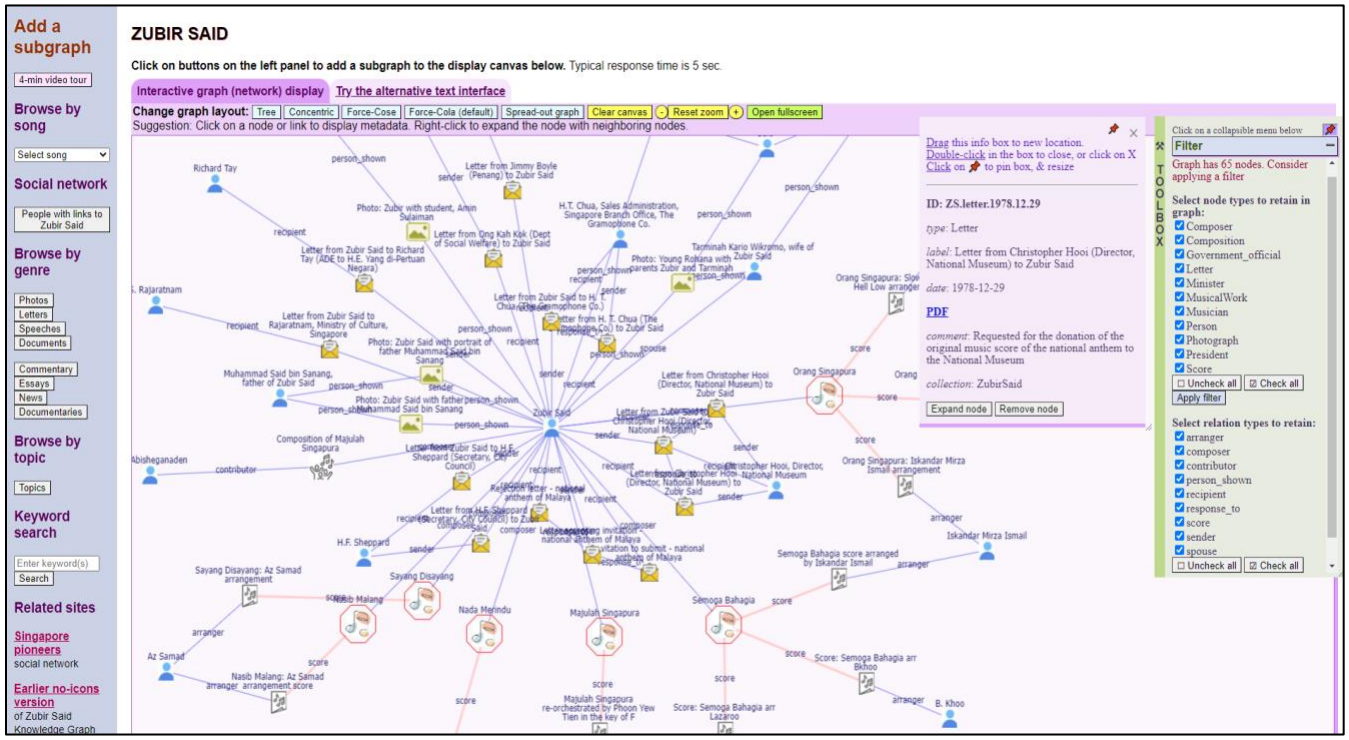
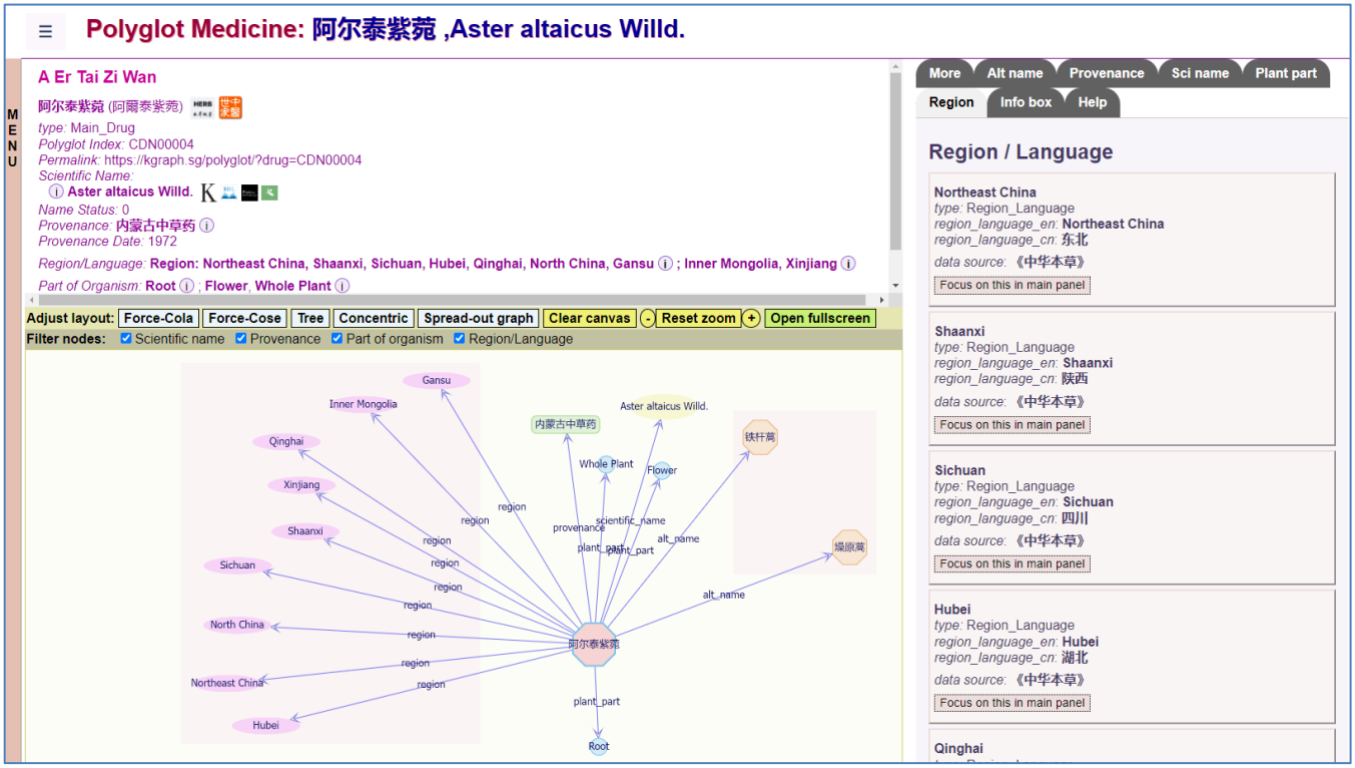


Figure 6. Polyglot Medicine web interface, showing information for one drug node in the main text panel, related nodes in tabbed text panels, a graph visualization, and a menu panel on the left (currently hidden, but slides into view on mouse-over).



Search Menu Panel

The search menu panels for the three web interfaces are shown in figure 7. The design of a search menu is linked to the design of the search queries, which depends on the application purpose of the system and the user needs/tasks that it purports to support. Most of the queries, including canned search queries associated with appropriately labeled buttons, will involve graph pattern matching (i.e., matching both nodes and links). They must be carefully crafted to retrieve a meaningful subgraph for display, but not too many nodes as to overwhelm the user. Crafting a query often involves deciding on the node types to display, the number of links (path length) to traverse (single link is usually the minimum), and the types of links to traverse. The user may supply one or more parameters to be incorporated in a graph query.

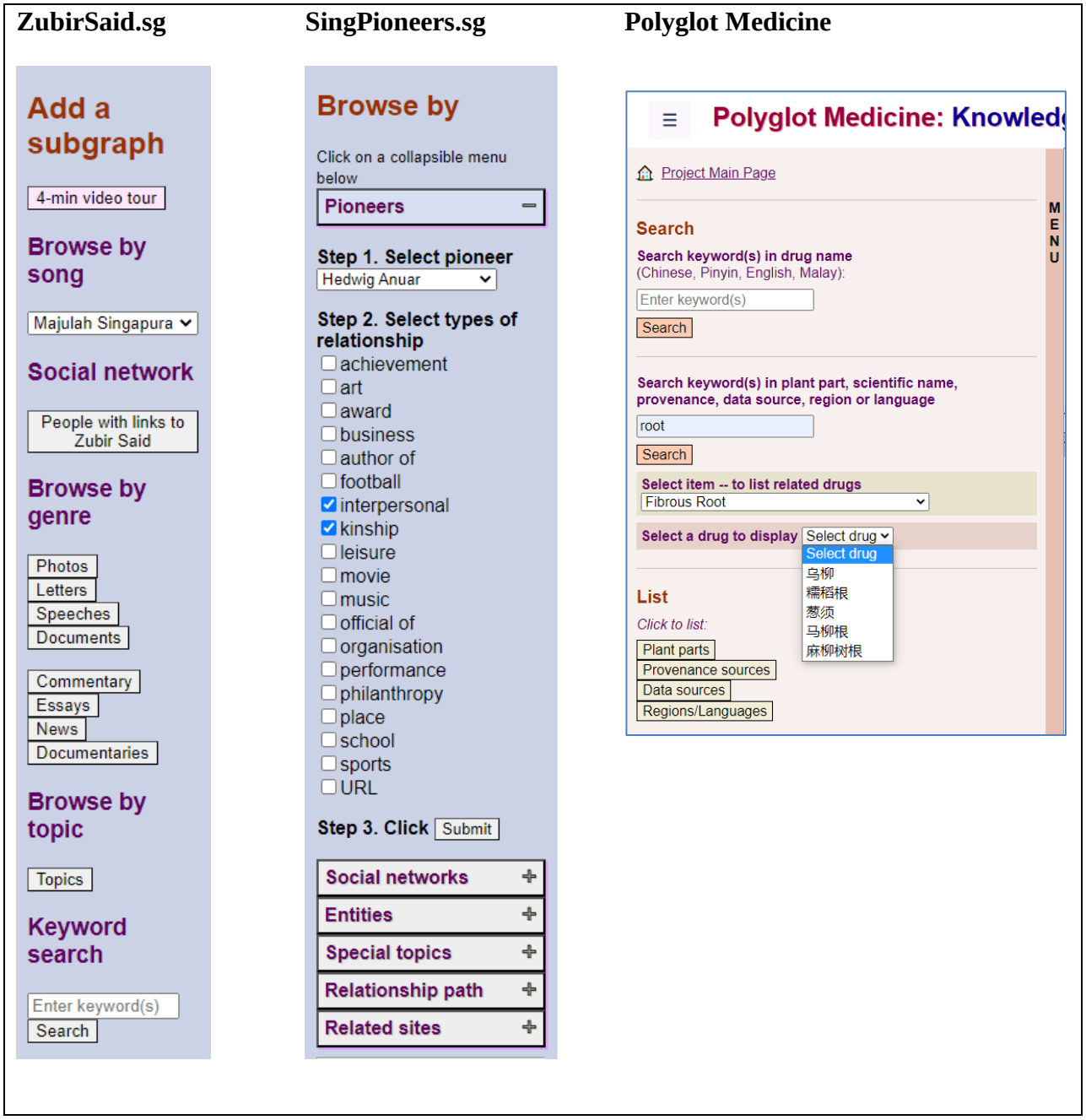
There are three main types of queries that can be offered in the search menu:

1. Thematic queries—these are usually canned queries that the user can execute by clicking on the associated buttons. Thematic queries are usually focused on particular resource genres such as photographs and letters, or particular topics, or types of social network.
2. Ego-centric or entity-centric subgraphs—for example, a subgraph centered on the person Zubir Said or the composition *Majulah Singapura*. The path length (i.e., number of links) of the related nodes to display will determine how many nodes are retrieved. A pulldown menu can list all the persons or entities for the user to select one as a parameter to be incorporated in a graph query. If there are too many entities to list in a pulldown menu, a keyword search box is needed to shortlist a smaller number of matching entities for the user to select. This is illustrated in the search menu for Polyglot Medicine (see fig. 7, right-

hand panel). Here, the keyword *root* (plant part) matches several types of *root*. Selecting *Fibrous root* retrieves five Chinese drugs for the user to select. The search menu of SingPioneers.sg (see fig. 7, middle panel) shows how checkboxes can be used to specify the link types to use for ego-centric searches.

3. Keyword search box—allows the user to enter keywords to retrieve a set of records (nodes) to display. In the ZubirSaid.sg interface, the keyword search retrieves individual nodes without links. This is so the user will not be confused about which nodes match the keyword.

Figure 7. Search menu panels from ZubirSaid.sg, SingPioneers.sg, and Polyglot Medicine.



There are other types of advanced graph queries that can be executed on a Neo4j database, based on graph pattern matching, graph traversal, and social network analysis.

The graphs retrieved and displayed by these queries are just starter graphs. The user can interact with the graph display and expand individual nodes with neighbor nodes that are linked to it. Submitting multiple queries (by clicking on multiple buttons) adds and merges multiple subgraphs to form a combined graph in the graph visualization. This allows the user to examine possible links between multiple subgraphs. This is different from typical search engine and database interfaces, where submitting a new query will generate a new search result display.

Graph Visualization Design

The graph visualization is what makes this interface special. As the graph visualization presents the search result of a graph query, the graph visualization should include text labels for nodes and links. Users are expected to read the text labels to figure out what entities/concepts/resources the nodes and links represent. The text labels play the same role as document titles in a bibliographic retrieval system. This is different from graph visualizations of large networks without text labels. Our knowledge graph displays are expected to contain fewer than 150 nodes, as we found that graphs with more than 150 nodes are unmanageable for information tasks.

Taylor and Rodgers provided a systematic treatment of aesthetic design principles for graph visualization, based on graphical design principles.[27] This section will focus on the following graph design elements: size of canvas display, node design, link design, use of icons, and layout algorithms (and their parameters).

The graph display canvas is sized relative to the size of the browser window so that users with big screens can display a big graph. A minimum size is specified so that on small devices the display canvas does not become too small (the user may have to scroll horizontally to view the rest of the canvas). We adopted a min-height of 400px and a min-width of 600px. Admittedly, graph visualization does not work well on small mobile phone screens; a text-based interface is recommended for such screens.

The design attributes of the nodes and the links (arrows) need particular attention. Different node designs are used to distinguish between the entity types that they represent. Design attributes for graph nodes include shape, size, color, the border of the shape, and display of node label. Important entity types can be made more prominent by using unusual shapes (e.g., octagon), a bigger size, and more saturated colors. Border colors and their attributes (e.g., thickness and type of line) can be used to indicate special attributes of the entity represented by the node.

Node design specifications for ZubirSaid.sg are as follows:

- Shape—ellipse (oval) as the default shape for the nodes. Different shapes are used for different classes; for example, octagon shape is used for Musical Work and rounded rectangle (like a picture frame) for Person. We limited the number of shapes used, as many different shapes make it more difficult to pick out a particular shape.
- Color—colors with lower saturation for the node background color (e.g., color saturation 80% and lightness 90% in the HSL representation). We recommend using the HSL (hue, saturation, lightness) color representation as it is easier to fine-tune the light/dark and saturation values of a color. Again, we limited the number of colors used, as too many colors are distracting. Each color is assigned to a group of classes (or a top-level class), rather than a different color for every class.

- Node labels—more space for displaying node labels, with a maximum width of 180 pixels and allowing the text to wrap around to three lines. This is to help the user to make sense of the resources represented by the nodes and links.
- Photographs on nodes—we considered displaying thumbnail photographs of persons in the nodes, but decided against it as it would slow down the response time and make the display look cluttered.

Von Landesberger et al. pointed out that link design is important.[28] Holten and Van Wijk found in their user study that while arrows were widely used, color transitions (from color A to color B, indicating the direction), and thickness transitions (from thick to narrow) are better.[29] The design attributes for the links include color, thickness, type of line, and type of arrowhead.

For ZubirSaid.sg, we did not use arrowheads in the graph display, as the direction of the relations is usually clear from the context. We learned from a previous project that arrowheads are obtrusive and distract from other graph elements such as the text labels. However, for SingPioneers.sg and Polyglot Medicine, we found it necessary to display arrowheads to indicate the direction of the relationship. We used a hollow vee as arrowhead (i.e., `target-arrow-shape: vee, target-arrow-fill: hollow`). We found that a hollow arrowhead (as opposed to a solid fill) makes the arrowhead less obtrusive and yet clear. As there are occasionally two or more links between two nodes, we use a bezier curve-style for the links, so that multiple links are visible and not superimposed.

We had initially not used icons to represent nodes, believing that icons would interfere with the node labels. However, so many participants of the user study recommended using icons that we reconsidered our decision. We found that colored icons do not distract from the node labels, provided their colors are carefully adjusted to be less saturated so as not to attract too much attention. The icons used in ZubirSaid.sg were carefully selected (from iStock.com) to be evocative of the node type. An unexpected advantage is that the icons can be small yet clear. As the node labels are in a different color, users can mentally filter out the text to focus on the network topology. Thanks to the variety of high-quality icon designs in iStock.com, the icons are, we think, quite successful in helping users to quickly grasp the type of resource represented by the node.

A useful feature in Cytoscape.js is the capability to assign a group of nodes to a parent node, thereby clustering the group of nodes. We found this particularly useful when an entity-centric node has many links to neighbor nodes of the same type, which can then be clustered together in the display. This is adopted in the Polyglot Medicine graph display, as shown in figure 6. When a main drug node is displayed, the *region* nodes are clustered together, as are the set of *alternative drug name* nodes. We applied this clustering to the social network graph shown in figure 5. The result of clustering letters, photographs, and music works (shown in fig. 8) makes the graph structure clearer.

The two-dimensional layout of the graph (i.e., positioning and distribution of the nodes and links) is an important factor affecting the readability of the visualization and allowing the user to examine and understand the "topological structure" of the graph.[30] Previous work on graph visualization design has focused on plotting a visually pleasing layout—what von Landesberger et al. called the "aesthetic criteria," such as "minimizing the number of [link] crossings, minimizing the total drawing area, and maximizing symmetries."[31] The focus of this paper is not on graph layout algorithm, which is technical in nature. Graph layout algorithms can be guided by adjusting user-defined parameters. So, our design decisions are at a higher level of abstraction: building on

existing good layout algorithms, we specify and adjust the available algorithm parameters to fine-tune the layout.

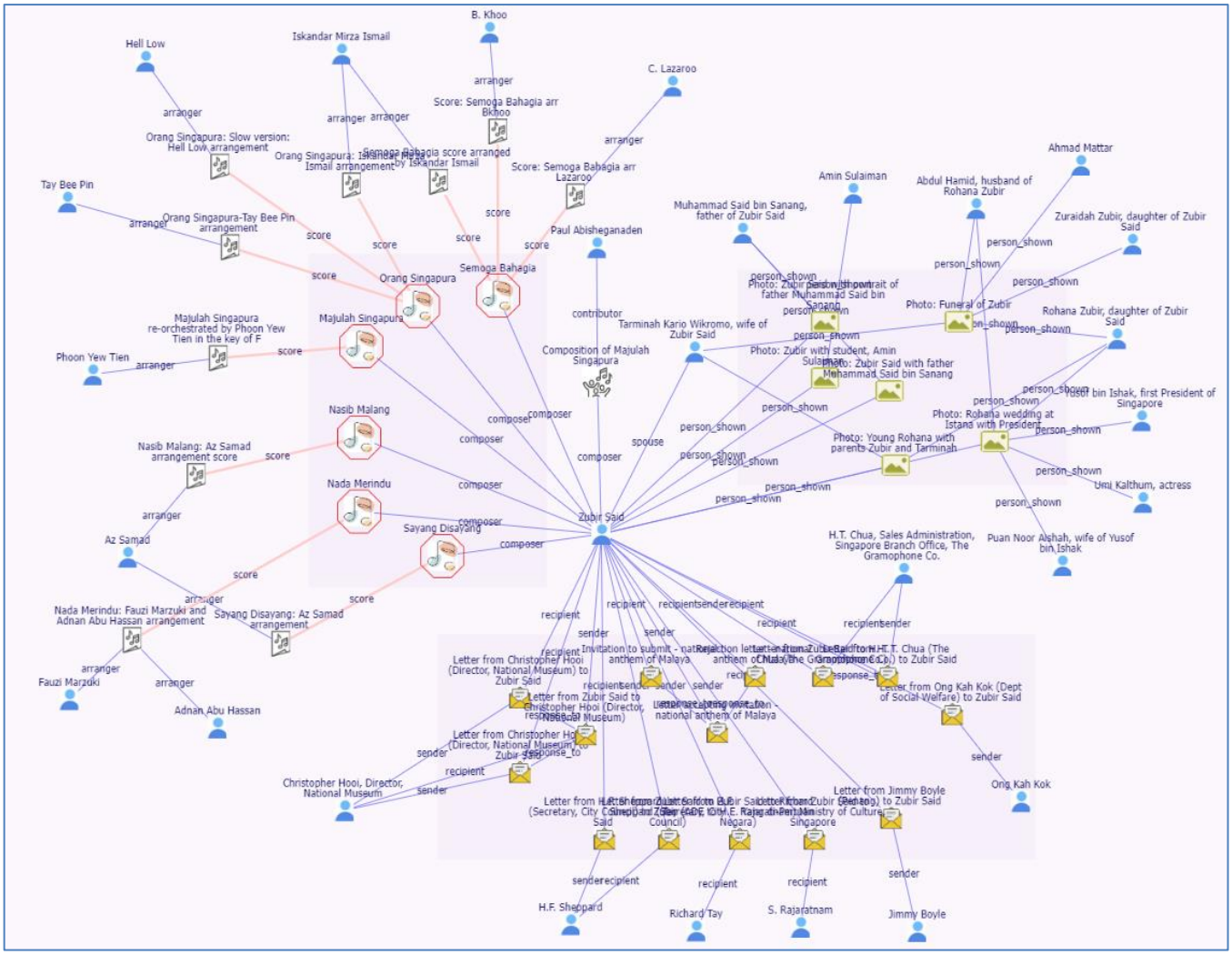
There are many types of layout algorithms. The main types are tree layouts, concentric layouts, matrix layouts, and force-directed layouts. Our focus is on force-directed layout algorithms based on physics simulations that balance repulsive forces between notes against attraction forces along the links. Force-directed layout algorithms are the most common type of layout algorithm used in graph visualizations.[32] Gibson, Faith, and Vickers and Cheong and Si have provided good surveys of force-directed graph layout algorithms.[33]

We adopted two force-directed layout JavaScript libraries that have worked well: Force-cola, and fCoSE.[34] fCoSE plots a good layout and attempts a different layout each time it is executed. However, we prefer Force-cola's dynamic layout where an animation of the layout shows how the algorithm incrementally adjusts and improves the layout. During the animation, the user can "pull" on particular nodes to help disentangle them and to guide the layout, as the algorithm will adjust the position of neighbor nodes in response to the user's node pulling. The disadvantage is that if there are many nodes retrieved, the initial (starting) layout looks messy and daunting, and it takes some time for the algorithm to gradually improve the layout. Our solution and recommendation is to apply both algorithms in sequence: first apply fCoSE to plot a reasonably good layout; then apply Force-cola to make further adjustments and provide an animation for the user to interact with.

All graph layout algorithms have parameters that can be adjusted to modify the algorithm and the graph layout. We recommend spreading out the graph more to improve the readability of the node and link labels, by increasing the node repulsion force (or node spacing) and also the length of the links (i.e., edge length).

The related issue of user-graph interaction is discussed in the next section.

Figure 8. A social network centered on Zubir Said, with links to other persons via intermediate entities (letters, photographs, music works, etc.).



Design of the User-Graph Interaction

The idea of using a graph visualization as an interface to a database or information system will be foreign to most users. The ZubirSaid.sg system offers a video tour of the interface to demonstrate the kinds of user-graph interactions that are possible. However, a more comprehensive help system is probably needed.

Von Landesberger et al. categorized graph interaction functions into three types:[35]

1. Change to the data—whether the user action affects the data displayed;
2. Change in visual form—whether the user action affects the parameters of the visual display or its visual representation; and
3. View interaction—including panning and zooming, "magic lenses," or fisheye view (distorting or changing the display to improve the readability of a small part of the graph).

We suggest another typology of user-graph interaction based on the number of nodes involved: operations on a selected node, operations on a small set of selected nodes, and operations on the

whole graph displayed. This typology can be combined with von Landesberger et al.'s categories to define nine types of interactions.

Operations on a selected node include repositioning a node by using the mouse (or finger touch) to drag the node to a different position, thus changing the visual form. This interaction is supported by most graph visualization software without additional coding effort. Operations on a specific node can also change the data displayed. Our interfaces allow the user to double-click or right-click on a node to retrieve from the database neighbor nodes that are directly linked to the clicked node. This requires writing JavaScript code to call a function provided by the Cytoscape.js JavaScript library to identify the ID of the clicked node, and then submitting a query (with the node ID) to the graph database. Our interfaces also allow the user to remove a selected node from the display by clicking on a *Remove node* button. A single-click on a node will display the associated node properties (i.e., metadata) on a text display panel. This will be discussed in the next section.

Operations on a small set of selected nodes are not yet implemented in the three interfaces under discussion. However, in other graph visualization applications we have worked on, additional functions have been added to allow the user to select two nodes and then request the graph database to identify the paths (or the shortest path) in the knowledge graph from one node to the other node. The identified paths are then added to the graph display. Another example of an operation on a set of selected nodes is to assign the multiple selected nodes to a super-node (i.e., parent node) to cluster them.

Operations on the whole graph displayed include trying different graph layout algorithms, which will change the visual form. Zooming in and out by using the mouse scroller button and panning the display canvas (viewport) will change the graph view. We recommend setting a minimum and maximum zoom level so that the graph does not become too small and difficult to locate in the display canvas. We use a minZoom value of 0.5 and a maxZoom of 4. A zoom reset is also needed to centralize the graph and to locate the graph should it disappear off the canvas and become lost in space. In addition to the operations that change the visual form and the view, filter menus (discussed later) allow the user to specify node types to remove from the display, thus changing the data displayed.

The graph interactions described above are basic interactions we expect to be supported in knowledge graph interfaces. More complex graph operations and graph queries can be coded and linked to various node events: clicking, double-clicking, right-clicking, mouse-over, and dragging (and the corresponding tap events on touch surfaces).

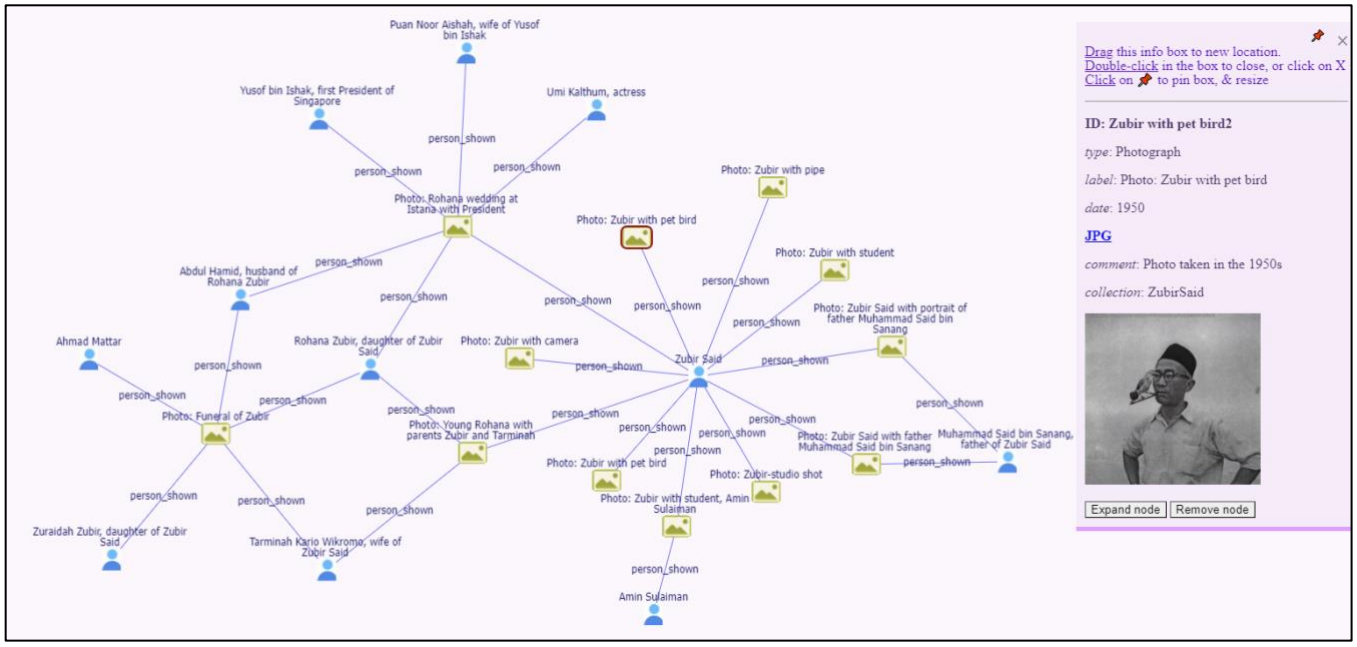
Text Display Panel

In the ZubirSaid.sg interface, a text display panel (info box) pops up when the user clicks on a node or a link in the graph display. This info box is important for displaying properties (metadata) associated with a node or link. As figure 9 illustrates, the info box also displays thumbnail images and hyperlinks when such information is stored in the node. Clicking on a thumbnail displays the corresponding high-resolution image. Clicking on a hyperlink displays an external webpage or a resource in the collection. Buttons are also provided in the info box to perform operations on the node. Currently two buttons are provided for removing the node from the display or expanding the node with neighbor nodes. The info box can be dragged and repositioned and also resized.

In the ZubirSaid.sg interface, the graph visualization takes center stage and is the primary display. The Polyglot Medicine interface takes a balanced approach of having both graph visualization and

text display (see fig. 6). The main text panel (in the center of the screen) displays properties for the main node in the graph display. The tabbed text panels on the right display information of the linked or neighbor nodes. The tab labels indicate the node type of the nodes displayed in each panel.

Figure 9. Graph visualization of photograph nodes and persons shown in the photographs.



Filter Menus

A filter menu is displayed on a sliding panel when the graph size exceeds 50 nodes (see fig. 10). We found in our user study that users are quite familiar with such filter menus, and some users spent a considerable amount of time working on it to control the nodes displayed.

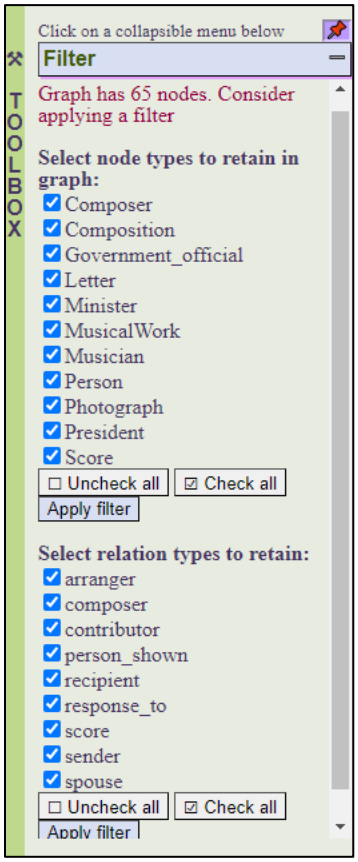
Our user study identified two unanticipated issues:

1. The filter menu displays both a node type filter and a link type filter. They interact in a way that users do not understand. A link cannot display independently of the two nodes that the link connects: a link will display only if the connected nodes are visible. For example, a spouse relation links two Person nodes. So, if the Person node type has been unchecked by the user and all Person nodes removed by the node filter, then the spouse relation will not display even if the spouse relation type is checked (selected) in the link type menu.
2. The node filter menu lists the node types as a flat list for the user to check/uncheck, without showing the class hierarchy. As the most specific class is assigned to each node, a node can be tagged as either Composer or Person, but not both. Users do not realize that Composer, Government_official, Minister, Musician, Person, and President listed in the filter menu are mutually exclusive. So, unchecking Composer but leaving Person checked means that composer Zubir Said will not appear in the graph display, even though Composer is a subclass of *Person*.

To avoid these types of confusion, we recommend that the interface offer just the *node type* filter. If a *link type* filter is desirable, then either the node type filter or link type filter should be active at any one time. For the Polyglot Medicine interface, we offer a short node type filter as part of the

menu bar of the graph display panel (see fig. 6). To avoid confusion from listing node types at different levels of the Class hierarchy, we recommend listing just the top-level classes in the filter menu.

Figure 10. Filter menus for node types and link types.



CONCLUSION

We have proposed a knowledge graph visualization interface as a new type of interface for digital heritage collections and, more generally, digital humanities-related datasets. It provides more support for users to study connections between resources (entities, concepts, and Web resources) and relationship structures—to facilitate browsing by navigating semantic links, information integration, and synthesis of a narrative understanding of an issue or topic. We have discussed the main issues in developing such an interface and offered recommendations based on our development experience as well as lessons learned from a small user study.

An interface design is not skin-deep: it must be supported by the underlying data and knowledge organization as well as the full stack of technology. In our system implementation, we have opted for lightweight and relatively inexpensive technologies that are effective, productive, and have enabled us to deliver good knowledge graph systems with a reasonable amount of effort and cost. We have recommended using the *labeled property graph* model for knowledge graph design, as a lightweight alternative to RDF/OWL2, and a graph database management system (available as a cloud service) that is based on this model. We have highlighted design issues in the graph visualization, as well as design issues in the knowledge graph, database, and web interface that have implications for the graph visualization and user interaction. We have also outlined the

technologies used in the workflow—starting with Google Sheets as the main data store for data entry, and Neo4j graph database management for search and analytic functions.

We believe that much more can be accomplished with knowledge graph and graph visualization technologies. Future enhancements to our interfaces include more advanced graph operations as well as graph analysis functions—for example, to trace relationship paths between two specified persons or entities, to identify pairs of entities with a specified relationship path between them (using graph pattern matching), as well as to identify constellations of entities participating in a particular relationship structure. Our ongoing work in developing knowledge graph applications include visualization of the argument structure of research papers (https://kgraph.sg/argstructure/demo1.html) and visualization of research results summarized in literature reviews (https://kgraph.sg/litreview/). These require more structured graph visualizations (with more precise placement of nodes) to help users understand the argument structure and compare research results across papers.

ENDNOTES

[1] E. W. Schneider, "Course Modularization Applied: The Interface System and its Implications for Sequence Control and Data Analysis" (paper presented at the meeting of the Association for the Development of Instructional Systems (ADIS), Chicago, IL, April 1972), https://files.eric.ed.gov/fulltext/ED088424.pdf.

[2] "How Google's Knowledge Graph Works," Google, https://support.google.com/knowledgepanel/answer/9787176?hl=en; Amit Singhal, "Introducing the Knowledge Graph: Things, Not Strings," *Google* (blog), May 16, 2012, https://blog.google/products/search/introducing-knowledge-graph-things-not/.

[3] Michael K. Bergman, "A Common Sense View of Knowledge Graphs," *Adaptive Information, Adaptive Innovation, Adaptive Infrastructure* (blog), July 1, 2019, https://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/; Aidan Hogan et al., *Knowledge Graphs*, Synthesis Lectures on Data, Semantics, and Knowledge, no. 22 (Berlin: Springer, 2021), https://doi.org/10.2200/S01125ED1V01Y202109DSK022.

[4] Lisa Ehrlinger and Wolfram Wöß. "Towards a Definition of Knowledge Graphs," (presentation in Posters and Demos Track at 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16), Leipzig, Germany, September 13–14, 2016), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1054.8298&rep=rep1&type=pdf.

[5] Jesús Barrasa, "RDF Triple Stores Vs. Labeled Property Graphs: What's The Difference?" (paper presented at GraphConnect, San Francisco, CA, October 2016), *Neo4j* (blog), 2017, https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/; Kevin Feeney, "Graph Fundamentals—Part 2: Labelled Property Graphs," *Medium*, TerminusDB Community (blog), 2019, https://medium.com/terminusdb/graph-fundamentals-part-2-labelled-property-graphs-ba9a8edb5dfe.

[6] "Defining N-Ary Relations on the Semantic Web: W3C Working Group Note," World Wide Web Consortium, April 12, 2006, https://www.w3.org/TR/swbp-n-aryRelations/.

[7] However, Neo4j recently introduced node property existence constraints in the enterprise version of the software (see https://neo4j.com/docs/cypher-manual/current/constraints/), which allows the developer to specify mandatory node properties.

[8] "Best Graph Databases," *G2.com* (blog), 2023, accessed June 2, 2023, https://www.g2.com/categories/graph-databases; "DB-Engines Ranking of Graph DBMS," *DB-Engines* (blog), 2023, accessed June 2, 2023, https://db-engines.com/en/ranking/graph+dbms.

[9] Min Wu, "Graph Database Market Overview," *Nebula Graph* (blog), 2023, accessed June 2, 2023, https://www.nebula-graph.io/posts/graph-database-market-overview.

[10] "What Is Cytoscape?," Cytoscape Consortium, accessed January 31, 2024, https://cytoscape.org/what_is_cytoscape.html; Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker, "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks," *Genome Research* 13, no. 11 (2003): 2498–2504.

[11] Max Franz, Christian T. Lopes, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D. Bader, "Cytoscape.js: A Graph Theory Library for Visualisation and Analysis," *Bioinformatics* 32, no. 2 (2016): 309–11, https://doi.org/10.1093/bioinformatics/btv557.

[12] "A Comparison of Javascript Graph/Network Visualisation Libraries," *Cylynx* (blog), March 9, 2021, https://www.cylynx.io/blog/a-comparison-of-javascript-graph-network-visualisation-libraries/; Elise Devaux, "List of Graph Visualization Libraries," *Elise Devaux* (blog), May 5, 2019, https://elise-deux.medium.com/the-list-of-graph-visualization-libraries-7a7b89aab6a6.

[13] Tatiana von Landesberger et al., "Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges," *Computer Graphics Forum* 30, no. 6 (2011): 1719–49, https://doi.org/10.1111/j.1467-8659.2011.01898.x.

[14] Vahan Yoghourdjian et al., "Exploring the Limits of Complexity: A Survey of Empirical Studies on Graph Visualisation," *Visual Informatics* 2, no. 4 (2018): 264–82.

[15] Weidong Huang, Peter Eades, and Seok-Hee Hong, "Measuring Effectiveness of Graph Visualizations: A Cognitive Load Perspective," *Information Visualization* 8, no. 3 (2009): 139–52.

[16] Yoghourdjian et al., "Exploring the Limits of Complexity," 264–82.

[17] The maximum possible links for a directed graph is calculated with the formula n * (n−1), where n is the number of nodes. This number is halved for an undirected graph, i.e., where the links do not have a direction.

[18] Yoghourdjian et al., "Exploring the Limits of Complexity," 264–82.

[19] Landesberger et al., "Visual Analysis of Large Graphs," 1719–49.

[20] Chih-Ming Chen et al., "Development and Application of a Digital Humanities Research Platform for Biographies of Malaysian Personalities," *The Electronic Library* 40, no. 4 (2022): 313–37, https://doi.org/10.1108/EL-01-2022-0007.

[21] Eero Hyvönen and Heikki Rantala, "Knowledge-Based Relation Discovery in Cultural Heritage Knowledge Graphs," in *Digital Humanities in Nordic Countries: Proceedings of the Digital Humanities in the Nordic Countries 4th Conference* (CEUR-WS.org, 2019), https://helda.helsinki.fi/bitstream/handle/10138/304630/21_paper.pdf?sequence=1.

[22] Junchao Wu et al., "The Canton Canon Digital Library Based on Knowledge Graph—Taking the Revolutionary Archives of Canton in the Republic of China as an Example," in *10th International Conference on Educational and Information Technology (ICEIT)* (Piscataway, NJ: IEEE, 2021), 171–79, https://doi.org/10.1109/ICEIT51700.2021.9375538.

[23] Xiaoguang Wang, Xu Tan, and Shengping Xia, "Dunhuang Wisdom Data Research and Practice" (in Chinese), *Digital Humanities in China,* no. 4 (2020), https://www.dhcn.cn/site/works/dhjournal/202004/6048.html.

[24] Jinhua Dou, Jingyan Qin, Zanxia Jin, and Zhuang Li, "Knowledge Graph Based on Domain Ontology and Natural Language Processing Technology for Chinese Intangible Cultural Heritage," *Journal of Visual Languages & Computing* 48 (2018): 19–28.

[25] Charlene Ellul, Joel Azzopardi, and Charlie Abela, "NotaryPedia: A Knowledge Graph of Historical Notarial Manuscripts," in *On the Move to Meaningful Internet Systems: OTM 2019 Conferences* (Berlin: Springer, 2019), https://doi.org/10.1007/978-3-030-33246-4_39.

[26] In the pulldown menu *File>Share>Publish to web.*

[27] Martyn Taylor and Peter Rodgers, "Applying Graphical Design Techniques to Graph Visualisation," in *Ninth International Conference on Information Visualisation (IV'05)* (Piscataway, NJ: IEEE, 2005), 651–56.

[28] Landesberger et al., "Visual Analysis of Large Graphs," 1719–49.

[29] Danny Holten and Jarke J. van Wijk, "A User Study on Visualizing Directed Edges in Graphs," in *Proceedings of the International Conference on Human Factors in Computing Systems, Boston* (New York: ACM, 2009), 2299–2308.

[30] Helen Gibson, Joe Faith, and Paul Vickers, "A Survey of Two-Dimensional Graph Layout Techniques for Information Visualisation," *Information Visualization* 12 no. 3 (2013): 324–57.

[31] Landesberger et al., "Visual Analysis of Large Graphs," 1723.

[32] Yoghourdjian et al., "Exploring the Limits of Complexity," 264–82.

[33] Gibson, Faith, and Vickers, "A Survey of Two-Dimensional Graph Layout Techniques," 324–57; Se-Hang Cheong and Yain-Whar Si, "Force-Directed Algorithms for Schematic Drawings and Placement: A Survey," *Information Visualization* 19, no. 1 (2019): 65–91.

[34] Hasan Balci and Ugur Dogrusoz, "fCoSE: A Fast Compound Graph Layout Algorithm with Constraint Support," *IEEE Transactions on Visualization and Computer Graphics* 28, no. 12

(2022): 4582–93; Ugur Dogrusoz et al., "A Layout Algorithm for Undirected Compound Graphs," *Information Sciences* 179 (2009): 980–94.

[35] Landesberger et al., "Visual Analysis of Large Graphs," 1719–49.