COS333 Final Project: Design Document

SAGE - The Smarter Academic Graph Explorer

Edward Zhang, Team Leader (edwardz) | Lehman Garrison (lgarriso) | Brenda Hiller (bhiller) | Connie Wan (cwan)

March 28, 2012

1 Overview

The body of academic literature is enormous. Performing research involves the daunting task of searching through this sea of works and determining which papers are relevant to a particular query. Furthermore, it is difficult to assess the relative importance of a paper - it is not obvious from reading a paper whether it is a seminal work introducing completely novel ideas, or an involved article with little meaning for those not already experts in the field.

The SAGE system helps researchers navigate through the space of academic works, allowing them to quickly find closely related works and locate important similar papers with ease. This is accomplished by visualizing the set of academic papers as a directed graph. Each paper is a node, and there is an edge from one paper to another if the first references the second. This formulation allows us to intuitively estimate importance with the heuristic of reference count - an important paper tends to be referenced many more times than less important papers. This heuristic is used by many academic search engines such as Google Scholar.

Most existing academic search engines are restricted to text-based keyword search. This makes navigating between related papers inefficient, since one cannot see the context in which each paper is operating. The graph model of academic literature is explored by Microsoft Academic Search (`academic.research.microsoft.com`) and the Web of Knowledge (`apps.webofknowledge.com`). These systems do not provide a way to navigate beyond the direct references of a single paper and are extremely slow to load because they aim to be complete. In contrast, our system does not need to display every single reference of a paper; this allows for rapid exploration of the graph and greater user-friendliness.

SAGE provides a basic keyword search to determine several relevant papers. These papers, and the references between them (if any), are displayed to the user visually. Our interface allows a user to center the graph around any of these papers, which will then display several of the referenced papers as well as some of those that reference it. The user can navigate around the graph by centering the view on a different paper, which will reveal new references. At all times, we keep the vital characteristics of relevance and importance visible to the user by means of a combination of size and color. When a node is clicked, we display more information about the selected paper in a sidebar and provide mechanisms for viewing the full text, saving the paper to a temporary portfolio to faciliate retrieval, and recentering the paper in the view of the graph.

SAGE is a web application that is based around the Microsoft Academic Search API. It uses a Django web server hosted on Google App Engine to interface between the client and Microsoft's API. The core graph visualization is based on the Arbor.js Javascript library (`http://www.arborjs.org`).

2 Functionality

Our basic mantra for the ideal user experience is to provide just enough information in just enough space. The goal is to have all the essential information already present, and any necessary information readily available. This also includes the ability for the user to adjust what is essential to her and to refine the way information is presented to make it more accessible.

2.1 UI Layout

Upon accessing our website, the user will be greeted by a clean title page reminiscent of Google's homepage. It will include our project title and a short, descriptive slogan. Beneath will be a search bar containing the text, "Enter author, title, subject, or keyword." The user also has the ability to refine search queries with an "Advanced Search" option located next to the search bar, where she can specify specific fields in which to search for names or keywords. At the top of the page will be a small menu bar with key links such as "About," "Contact," and "Help," which will remain on the page throughout.

2.2 Graph View

Once the user has entered a search query and chosen "Graph View", she will be taken to a page that displays the search query, still in the search bar, on the upper left. Front and center will be the graph that is the dominant visual feature of our project. A loading icon will appear while a graph of the initial search results is being generated. The user will then be presented with a graph centered on a vertex labeled as the current search query. This vertex is surrounded by large vertices labeled with the primary author and publication year of relevant papers, which are connected by a directed edge if there is a reference link between the papers. Farther away from the search-query vertex, vertices become smaller or change in tonal intensity to indicate diminishing relevance.

2.3 Graph Interaction

To navigate the graph, the user can click and drag the graph itself, or double-click on a vertex to center the graph on that vertex (and bring focus on that paper, as described below). Double-clicking on an edge will center the graph on the vertex to which it points, and edges to vertices currently not visible in the graph will still be displayed. A "home" button in the corner of the graph will re-center the graph on the starting vertex, which will be indicated by its color. The user will also see a "magnifying-glass" effect while navigating the graph, whereby vertices closer to the center of the graph are enlarged slightly and spaced more widely, for clarity. Once the user has identified a primary paper of interest from the preliminary search results, she may choose to focus on a "neighborhood" of that vertex using the viewing pane described below. The graph will then zoom in on that paper and center on its vertex, displaying one or two tiers of referenced and backreferencing papers around it (as indicated by directed edges). The "Home" button will now take the user back to the paper of focus. From here, the user may then explore this and other neighborhoods, or enter a new search query to restart the process.

Throughout the user's interaction with the graph, she may hover over or click on any single vertex for further information about that paper. By default, only the primary author and year of publication are displayed in each vertex on the graph. Hovering over a vertex will display a balloon with details including the title of the paper, all author names, the year of publication, and the number of forward and back references. Clicking or double-clicking on the vertex will bring up a tab in a viewing pane below the graph that displays a table of information about the paper, including a snippet of the abstract. A section within the pane will also allow the user to perform several actions, including the following:

- Rehome: the current vertex will become the home vertex, and will be colored appropriately.
- View Neighborhood: if the user is examining the preliminary search results, she will then be taken to the neighborhood view of the current vertex.
- Flag: the current tab will be pinned to the viewing pane, and the user will have the option of customizing the tab title. The vertex and the tab will both become colored, and an icon of that color will appear next to the Home button, allowing the user to jump to the neighborhood of that paper at any time.
- Unflag: an existing flag will be removed and the tab will be unpinned. Clicking on a different vertex will replace the current tab, rather than create a new one.
- View Paper: a new browser window will be opened displaying the paper. The graph will remain open in the original window.
- Go to Paper: the user will be taken away from the graph and redirected to the paper, at which point she must reinitialize her search to return to the graph.

The viewing pane itself may cover a portion of the graph when displayed, but clicking on an arrow in its title bar (or double-clicking the bar itself) will cause it to retract. If there are any pinned tabs, only the colored tab titles will be visible at the bottom of the screen. Otherwise, the viewing pane will retract fully so that only its title bar, with an arrow for expansion, is visible. The user may also lock the viewing pane in its retracted position with a lock icon in the corner of the title bar, in which case clicking on a new vertex will not automatically cause the viewing bar to expand. Manually expanding the viewing pane will unlock it from this position.

The graph is designed to be navigated in a simple and intuitive manner which allows for easy, quick navigation to the papers that are of the most interest to the user. As such, it will not have a panoply of displays and options, but rather a clean interface that highlights what is important and allows for the kind of focus needed to enhance the user's search. With these straightforward but highly functional tools, the user can then search efficiently and effectively for what she believes is the most important.

3 Design

3.1 System Architecture

The overall structure of SAGE is that of a client-server website that makes periodic requests of the server in reponse to user input. This server in turn queries the Microsoft Academic Search API, processes the results, and responds to the client. The design of the system makes the client responsible for most of the processing; the server essentially routes and translates requests from the client to Microsoft's servers. The server will be implemented in Python using Django on Google App Engine for ease of setup. The Django server may also implement basic user account features, possibly via Google accounts since the hosting is done on the Google App Engine. The most important feature will likely be saving graph views for later viewing.

The client interface is presented as a webpage in a brower, as described in the Functionality section. The main component of this interface is the graph representing papers and references, which will be constructed from lists of references. The graph will be managed entirely on the client using the Arbor.js library. Arbor.js is a Javascript graph visualization library, with several features (such as dynamic updating) that make it well-suited to SAGE. The client will also have several additional components dealing with viewing individual paper metadata. To facilitate development and interaction, SAGE will make heavy use of jQuery.

3.2 Queries and Interface

A user can perform two general actions: a keyword search, or selecting a node. The former produces a node representing the query, whose neighbors are the top resulting papers by relevance. The latter causes the graph to recenter on that paper, hiding distant nodes and displaying new, nearby nodes.

These two actions represent the two basic types of requests that the Django server will respond to. In the **search query**, the Django server will make a Microsoft Academic Search API call (using JSON) to retrieve a list of search results, with associated metadata such as relevance to the search term and number of incoming and outgoing references.

When zooming in on a node, the client performs a **neighbor query**. The Django server will query MSAS to get a list of papers that reference and are referenced by the current paper, as well as link counts and relevance. The client can then make additional **metadata query** calls to retrieve the metadata for these papers, such as author, abstracts, full text, and journal.

These query types can be easily translated into Microsoft Academic Search API calls. The server will perform this translation and make analagous calls to Microsoft's API, and then filter and retranslate the results.

In both cases, our Django server will limit the number of results it returns to the client based on the importance of the papers, so as to not transfer information that the client will not display anyway. Additionally, when a node is hidden based on its distance from the current paper, the node will be retained in memory in case the user revists it. Prefetching nodes that the user is likely to visit but are not yet displayed will probably be important for performance as well.

4 Milestones

Week 1 Mar 16: Design document, web site, elevator speech completed. Git repository on BitBucket set up.

Week 2 Mar 23: Basic App Engine web server installed. Client skeleton in place, including CSS, search page, display page (with placeholders).

Week 3 Mar 30: MSAS queries functional (keyword search, paper metadata, reference query). Client-Server interface code in place

Week 4 Apr 6: Dynamic graph functionality implemented - Navigating around graph, loading neighborhoods with AJAX calls.

Week 5 Apr 13: Prototype - Individual paper metadata view - e.g. authors, abstract, full text, etc. in separate pane

Week 6 Apr 20: Layout, CSS, and site design finalized (no major visual changes after this point). Finalize major feature set.

Week 7 Apr 27: Alpha Test. If all goes well, implement prefetching.

Week 8 May 4: Beta Test. Prepare demos, documentation, and presentation.

Week 9 May 11: Presentations. Final bug fixes.

Week 10 May 15: Final submission.

5 Risks and Open Issues

- Latency: The most significant risk is that of latency. Our system has two network-reliant communications channels: from the client to the server, and from our server to the Microsoft servers. To be able to navigate smoothly in the academic graph requires reasonably fast response times.

  To remedy this, we have several options. If the delay is significant, we would implement some form of caching or prefetching. This would most likely occur on the client side, although we certainly could perform prefetches and store them on the server as well. If we find our data is high in volume, we could also compress queries or eliminate/defer loading unused information.

  If the Microsoft Academic Search query times are far too long, then at an extreme we may have to store a subset of the graph on our own databases. Although this might be a good long-term performance solution, it is unlikely that we will have the time to do this in the duration of this project.

- A small risk is that our Microsoft Academic Search API application has not yet been accepted; if we are refused then our primary data source is lost.

- Many of our group members have very limited experience with Django and Javascript. The learning curve might impede our progress.