Conference-classifying Intergalactic Rationalization-retrieval (CIR)

CS3219 Assignment 5

Group 7

Apoorva Ullas, A0141138N

Mart van Buren, A0174937N

Valentin Gilbert, A0174830E

Loïc Vandenberghe, A0174721H

Github Repository

https://github.com/apoorva17/CS3219-A5

Introduction

For this assignment, we have built a web-based Information Retrieval Application which allows users to query and visualize the first 200,000 lines of data from the Open Research Corpus: Public Datasets of Scholarly Research Papers.

Vandenberghe Loic and Mart van Buren worked on the queries while Valentin Gilbert and Apoorva Ullas worked on the visualizations.

Requirement Specification

ID	Requirement	Type	Priority	Iteration
1	The system accepts intuitive user queries	Functional Requirement	High	1
1.1	The system uses default values when user selects trend from navigation bar dropdown	Functional Requirement	Medium	2
1.2	The system prompts user to provide input when there are empty input fields	Functional Requirement	Medium	2
1.3	The system displays error message “No Data Found For Query” for invalid user inputs	Functional Requirement	Medium	2
1.4	The system displays minimum/maximum valid year when user inputs value of year beyond the valid range	Functional Requirement	Medium	2
1.5	The system provides drop down	Functional	Medium	2

	options for attribute inputs	Requirement		
1.6	The user can show or hide form for a particular trend by clicking on the Trend Button for particular trend on homepage	Functional Requirement	Low	2
2	The system displays a visualization corresponding to user query	Functional Requirement	High	1
2.1	The user can hover over the visualization to view specific data in a popup dialog	Functional Requirement	Medium	1
2.2	The system displays a legend for the visualization	Functional Requirement	Medium	1
2.3	The user can change input for a query from the same page as the visualization	Functional Requirement	Low	2
3	The system should load in no more than 3 seconds	Non-Functional Requirement	High	1
4	The system should respond to user queries in no more than 5 seconds after clicking "Submit"	Non-Functional Requirement	High	1
5	The system should have an easy-to-use interface	Non-Functional Requirement	High	1
6	The system should support at least 5 types of user queries	Non-Functional Requirement	High	1
7	The system should support at least 3 types of visualizations	Non-Functional Requirement	High	1
8	The system should be able to store at least 200,000 lines of data	Non-Functional Requirement	High	1
9	The system should be able to handle exceptions due to invalid user input	Non-Functional Requirement	Medium	2
10	The system should not be vulnerable to web-based attacks	Non-Functional Requirement	Medium	2

11	The system should not use relational databases to store data	Non-Functional Requirement	Medium	1
12	The system should store data in JSON format	Non-Functional Requirement	Medium	1
13	The system should use the Model-View-Controller (MVC) framework	Non-Functional Requirement	Medium	1
14	The application interfaces with Google Charts to create the visualizations	Non-Functional Requirement	Medium	1
15	The source code should be open-source and readily available	Non-Functional Requirement	Low	1

Design and Implementation

Architecture

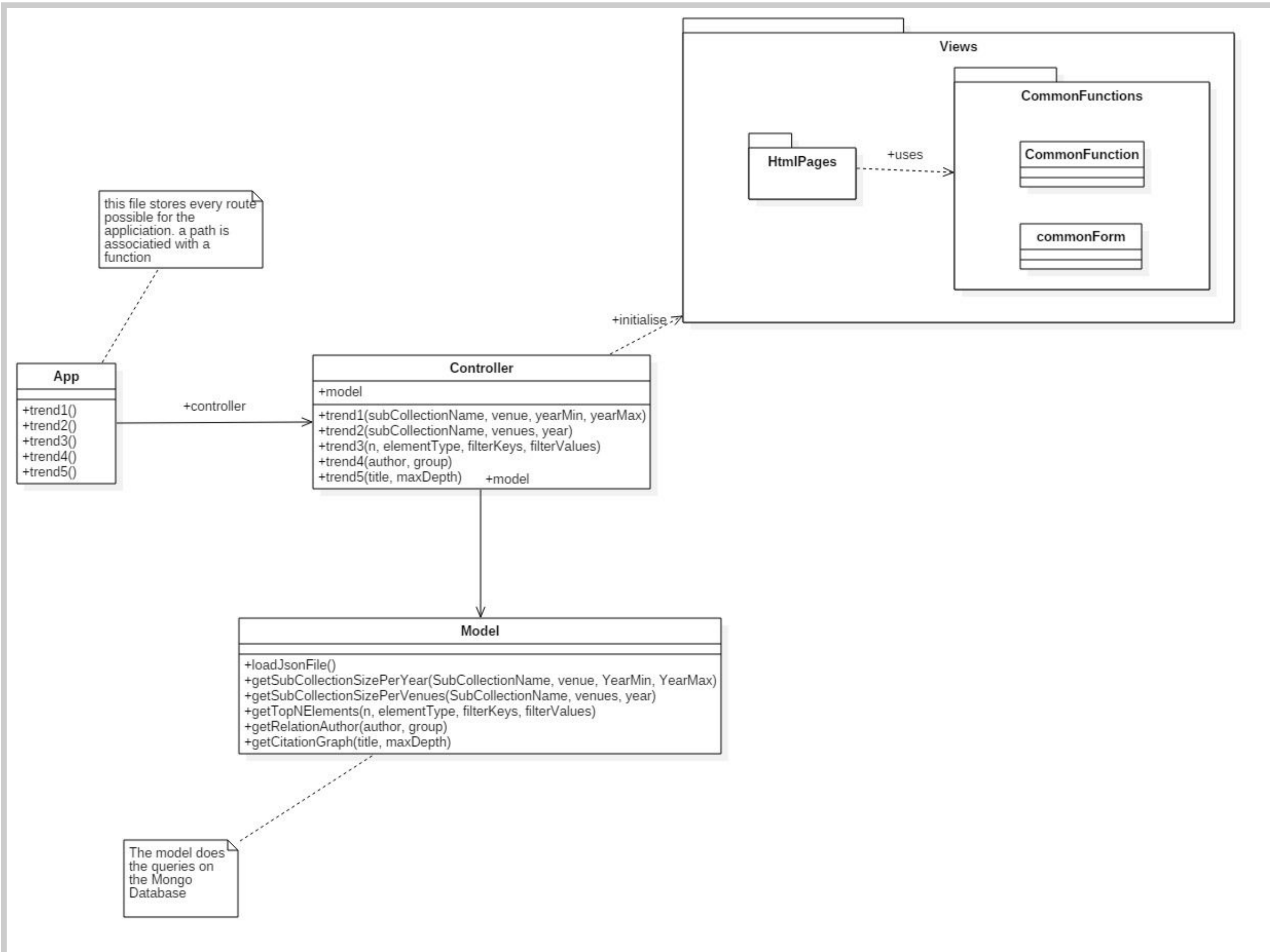

Figure 1: Architecture Diagram

Our application is based on the Model View Controller (MVC) architectural pattern principle.

We chose to implement the MVC pattern so that the View and Model can be completely independent. This increases the reusability and evolution of the software.

The App component stores all the routes of the application. Each time an URL is used, the request goes through the App component. This part of the application acts similar to a Front Controller as this is the only point of access to the application. The App component then build the controller to have access to the specified function requested.

When launched for the first time, the App component creates a controller. For each request, it calls the corresponding controller method of the request. The controller then analyse the arguments of the request and if necessary calls our API stored in the class Model. This class execute queries on a Mongo Database to collect the data corresponding to the user input. When the query is executed, the controller calls and initialises the corresponding view. Views are stored in the package htmlPages. Those views use the package CommonFunctions to display common contents like tool-bar or footer.

Design Principles

Our application applies the Separation of Concerns (SoC) design principle. It has been split into 4 main components: App, Model, View and Controller. Each component addresses a separate concern, with minimum overlap in functionality. For example, the View component specifically handles the user's interactions with the application. Applying such a design principle ensures modularity by encapsulating information inside a section of code which has a well-defined responsibility. This ensures that changes to one component does not affect the usability of other components. Hence, adding an additional query should not affect the functionality of other queries currently implemented, and neither should it require any modifications to the code for the other queries.

Technologies Used

As we were using the json file, we decided to create a MongoDB and load this json file into the database. This solution was very fast to implement and we could import any json file because the database is created dynamically. MongoDB is also a very fast tool which indexes its content, allowing us to have faster request on the data.

The model, controller and app are coded with Python. We used Python as our programming language as it is quick to get started, provides many useful libraries and allows for rapid development and prototyping.

We used Python Flask, a Python based web server which takes minimal time to setup and use.

The views are using Python Jinja2 to display HTML. We chose this as it works seamlessly with Flask and allows easy injection of data into HTML files.

Charts and graphs are generated by Google Chart and Vis Libraries. We chose Google Charts as it uses simple Javascript and simple to use libraries to quickly create our visualizations.

Lastly, we used Bootstrap and JQuery frameworks for the design of the views.

Trends

Trend 1: Transition over Time

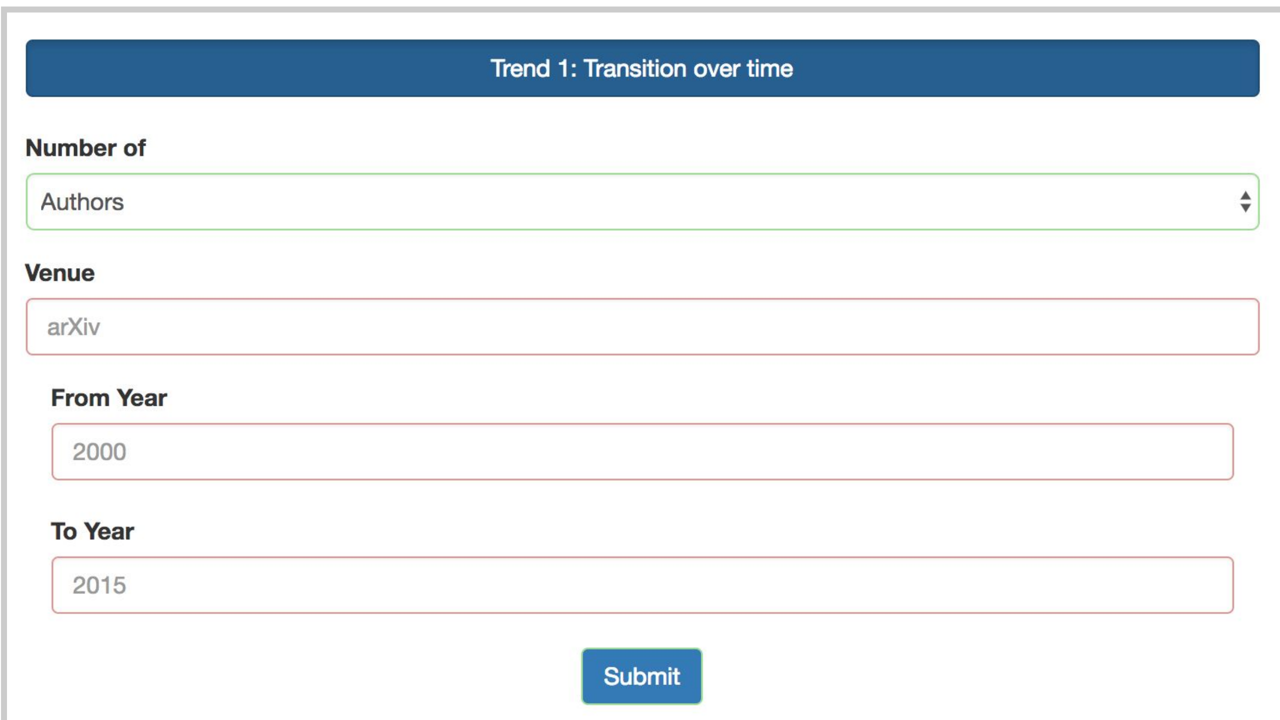

Figure 2.1: Query for Trend 1

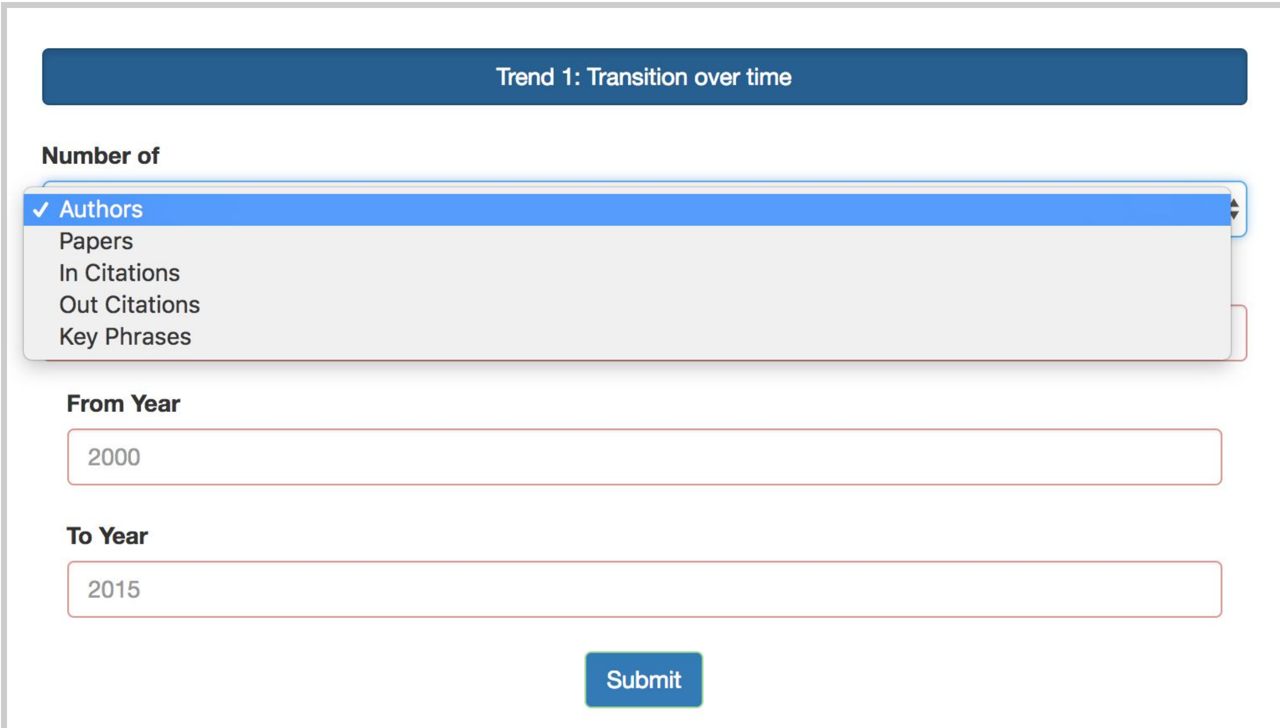

Figure 2.2: Dropdown option for Trend 1

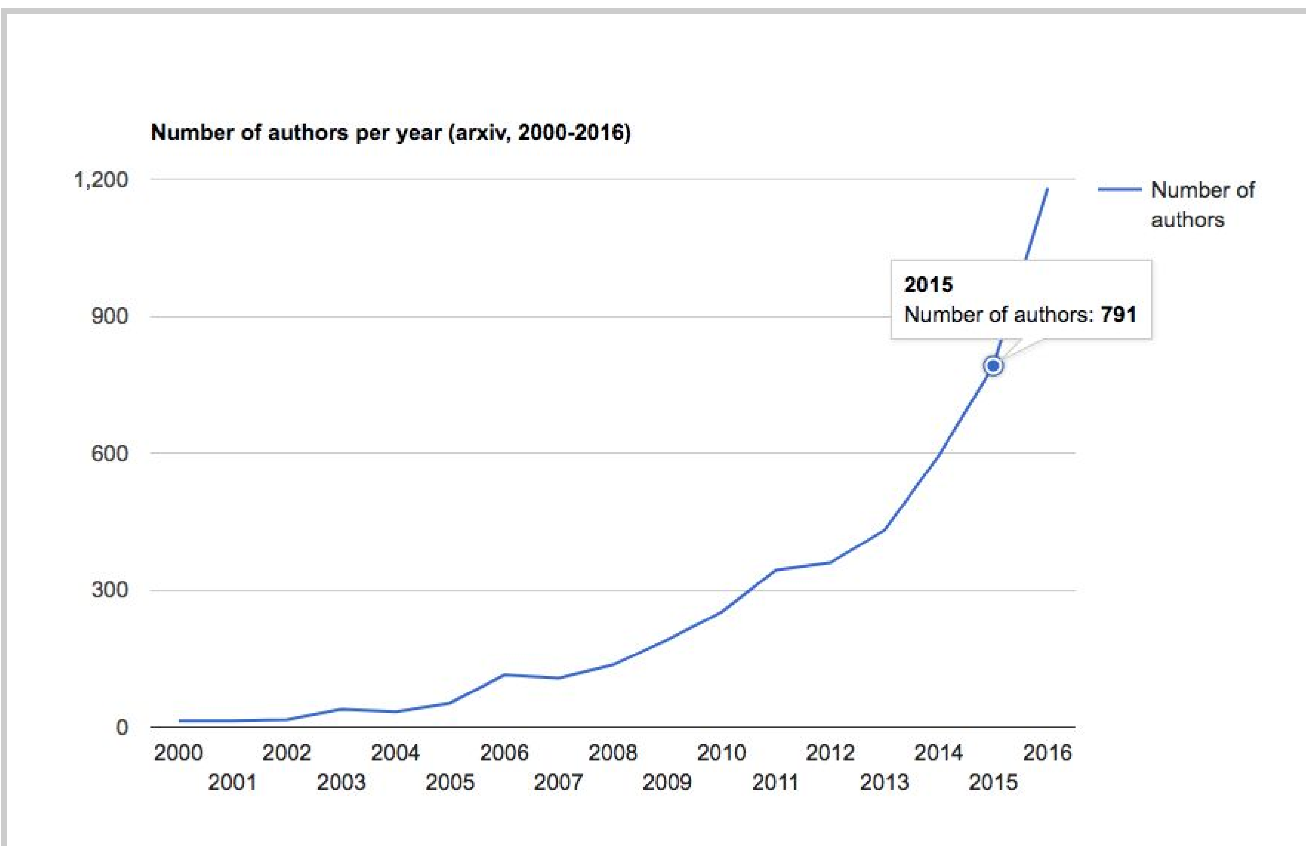

Figure 3: Line Chart Visualization

The purpose of this visualization is to see how a certain trend varies over time. The graph above is a line chart showing how the number of authors has changed from 2000 to 2016, for the venue "arXiv". Some common trends that can be viewed with this visualization are: the number of authors, number of papers, and number of citations.

Stepwise pipeline for generating this visualization:

- First filter for all papers with the provided venue and date range.
- Group all unique authors by year (in MongoDB: unwind the "authors" subcollection and group by "year").
- Sort by "year" in increasing order.

API:

1. Default request from navigation dropdown
   GET `/a5/trend1`
2. Request with user input parameters (sample request for above visualization)
   GET
   `/a5/trend1/?subCollectionName=authors&venue=arXiv&yearMin=2000&yearMax=2016`
   All parameters are required.

Trend 2: Contemporary Comparison

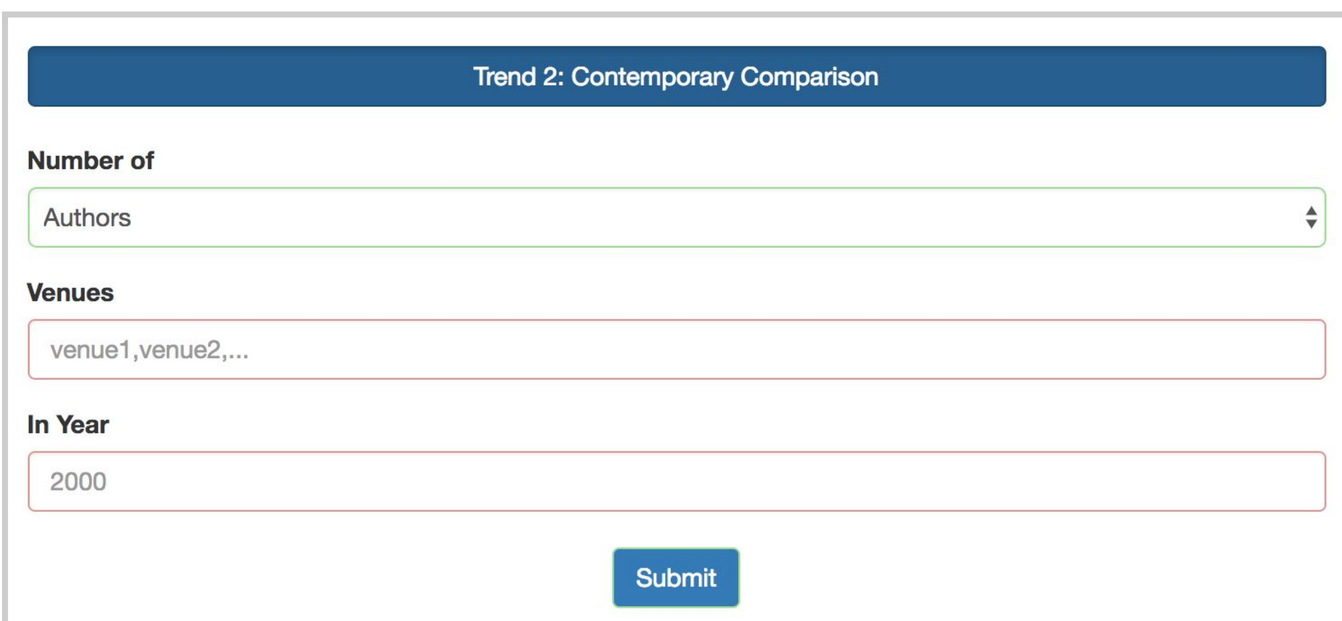

Figure 4.1: Query for Trend 2

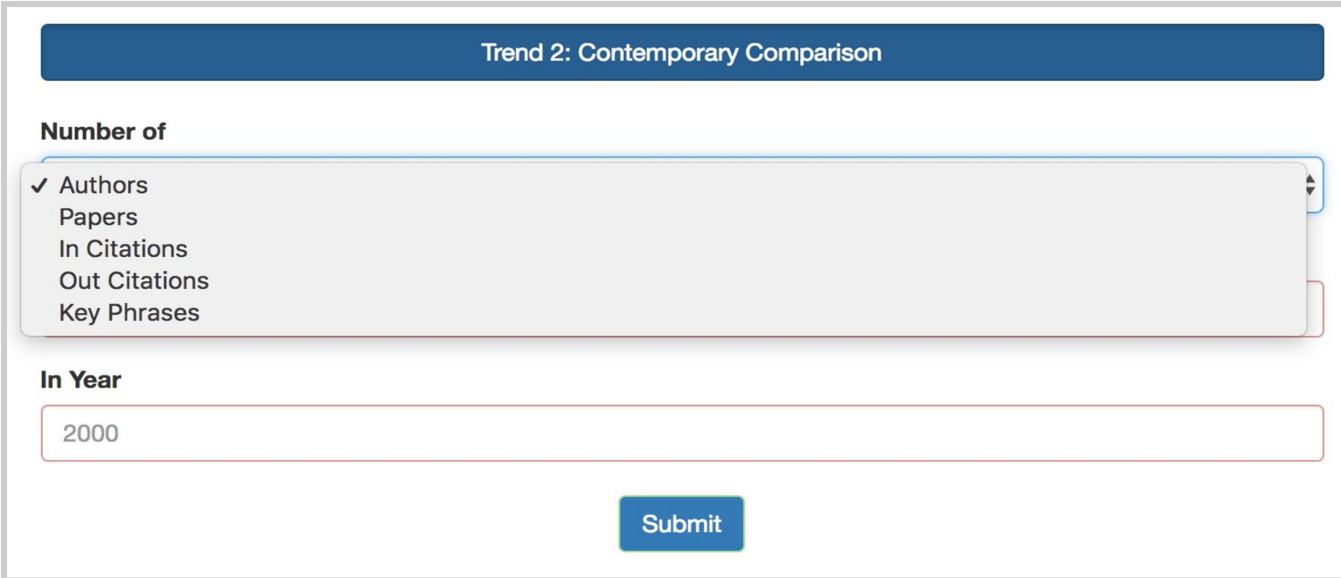

Figure 4.2: Dropdown for Trend 2

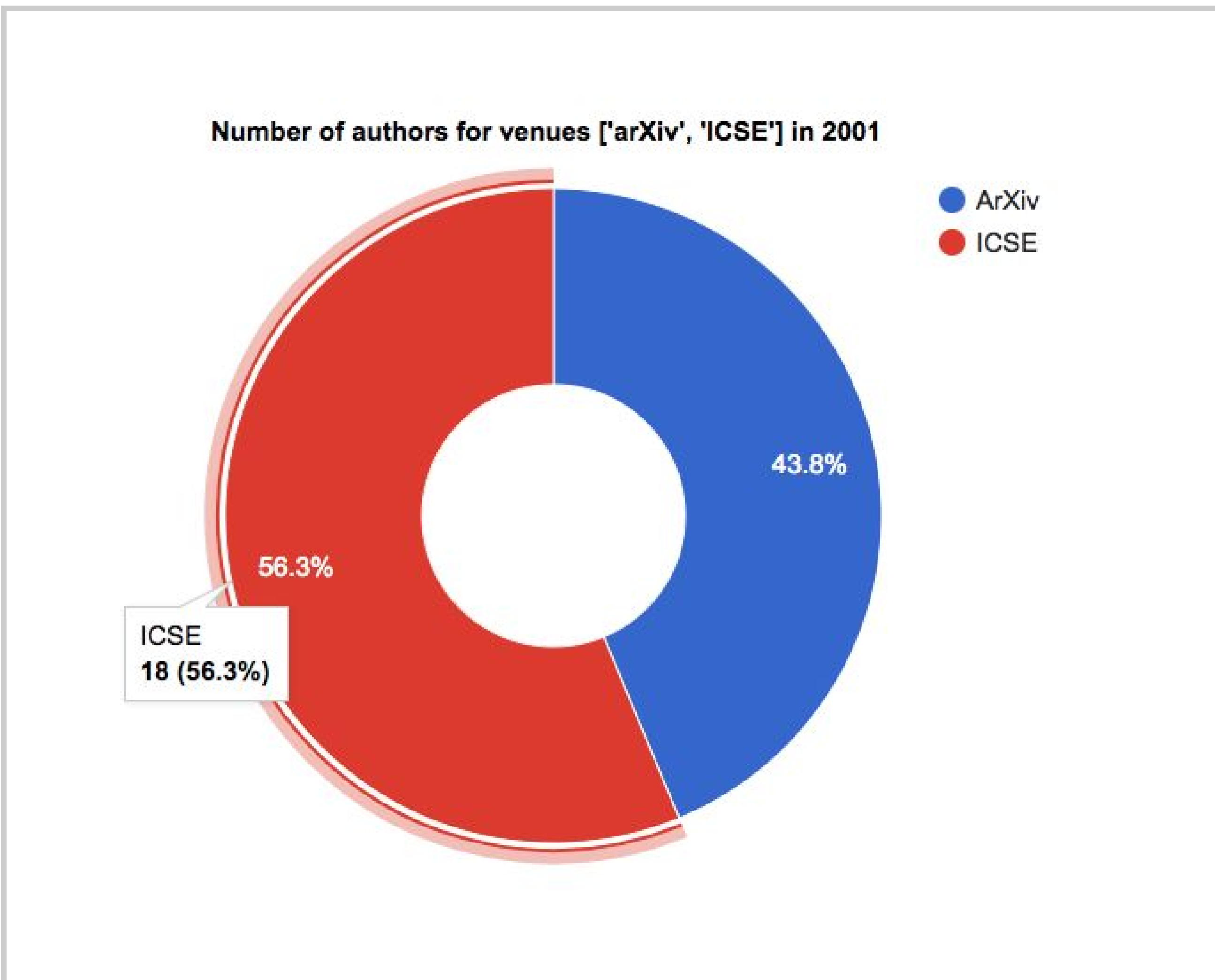

Figure 5: Pie Chart Visualization

The purpose of this visualization is to see how a certain trend varies between venues. The graph above is a pie chart showing how the number of authors varies between venues "ICSE" and "arXiv", for the year 2001. Some common trends that can be viewed with this visualization are: the number of authors, number of papers, and number of citations.

Stepwise pipeline for generating this visualization:

- First filter for all papers from the provided venues and year.
- Group all unique authors by venue (in MongoDB: unwind the "authors" subcollection and group by "venue").

API:

1. Default request from navigation dropdown
   GET `/a5/trend2`
2. Request with user input parameters (sample request for above visualization)
   GET `/a5/trend2?subCollectionName=authors&venues=arXiv%2CICSE&year=2000`
   All parameters are required.

Trend 3: Top N X of Y

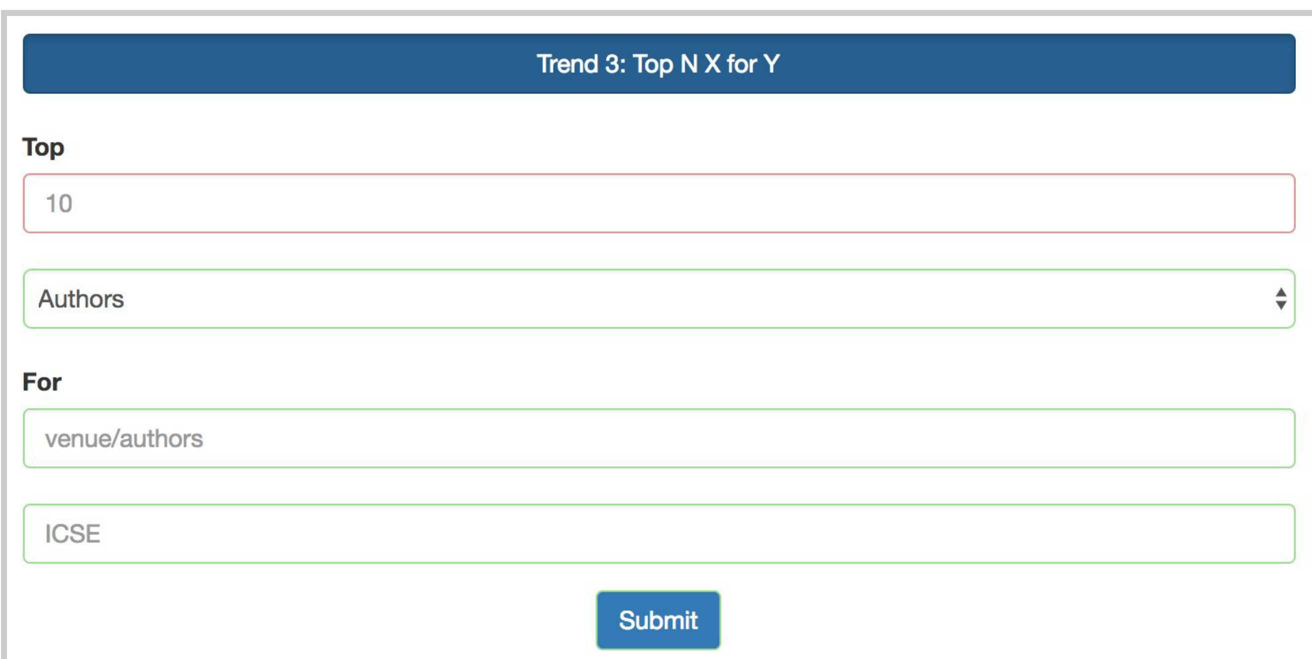

Figure 6.1: Query for Trend 3

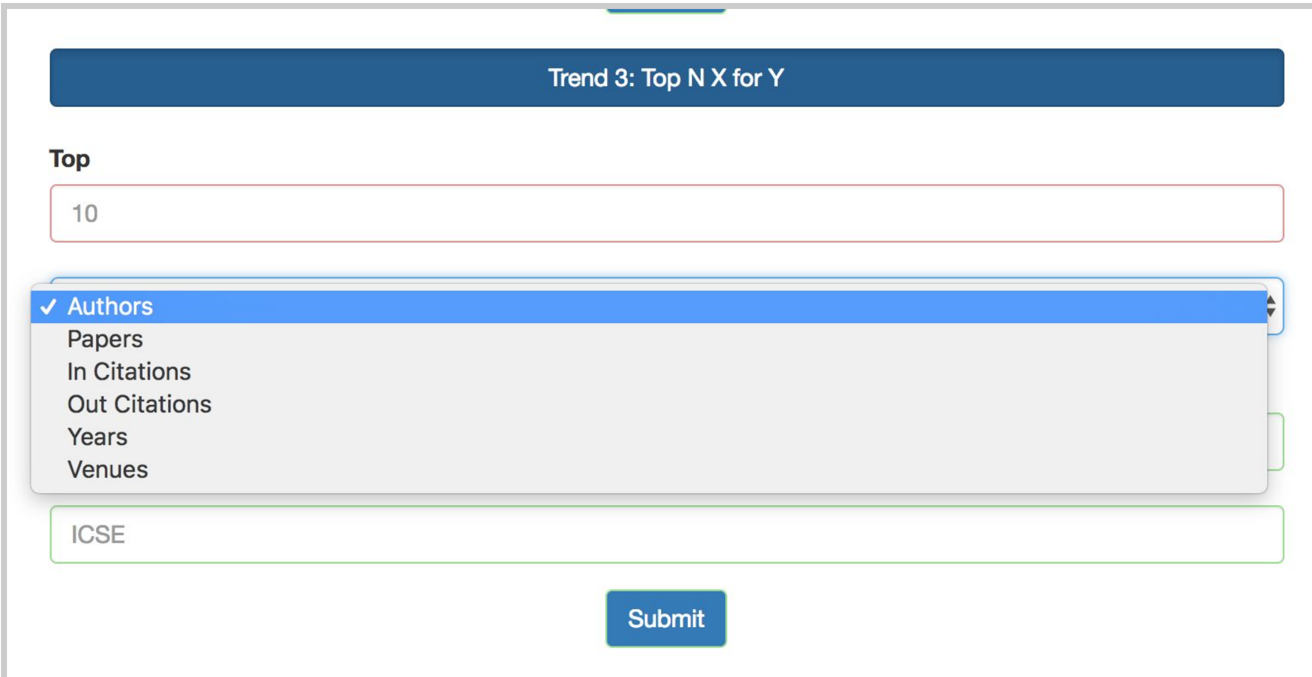

Figure 6.2: Dropdown for Trend 3

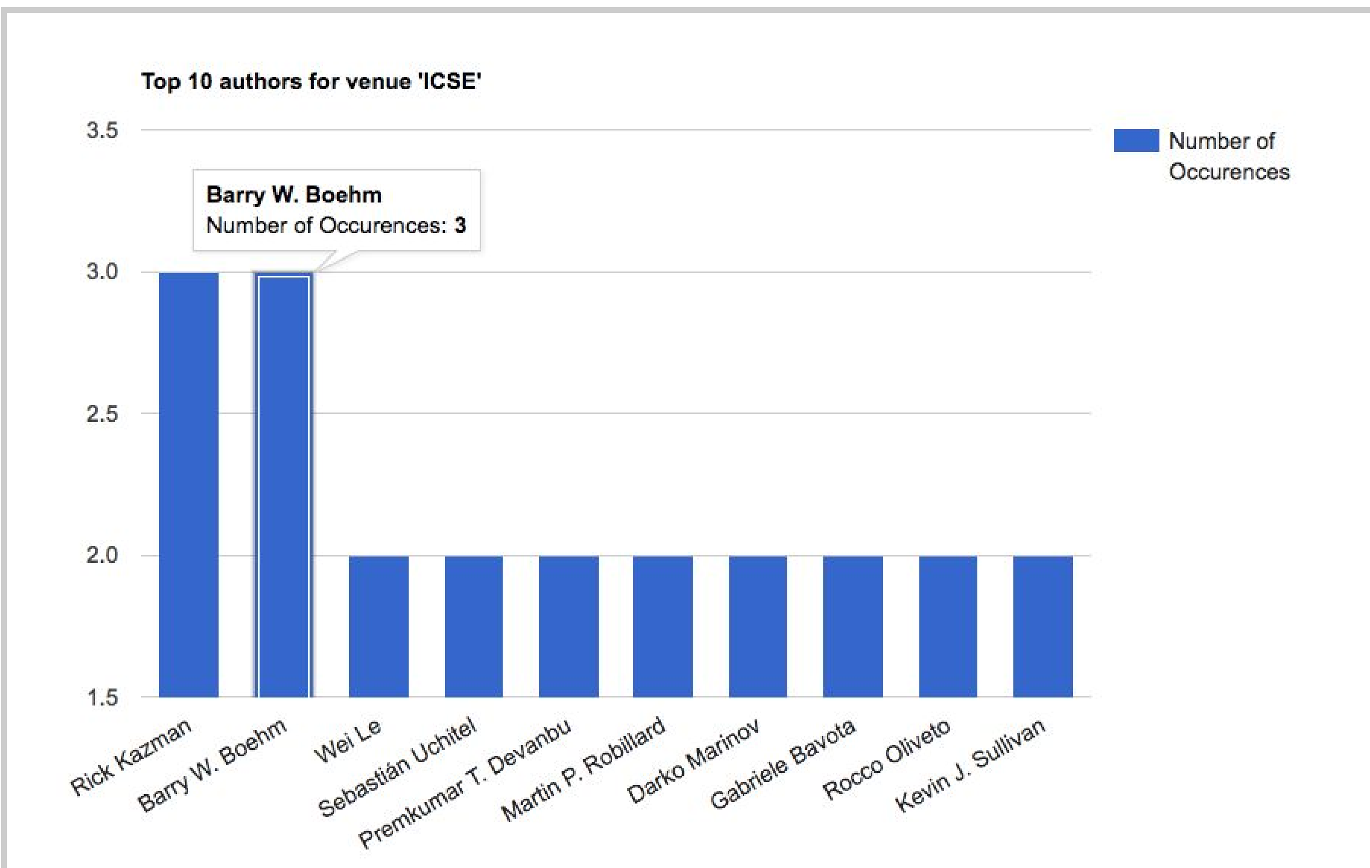

Figure 7: Bar Chart Visualization

The purpose of this visualization is to find the top N items of a certain category, subject to certain filters. Filters can be provided as regular expressions; this is makes it incredibly versatile. Some examples of what's possible: "Top authors for ICSE in 2001?", "Top venues in 2010?", "Top years for authors with first name Benjamin?", "Top years for papers starting with a 'W'?".

Stepwise pipeline for generating this visualization:

- (MongoDB detail) Expand all subcollections we are interested in. For example, if we are filtering by author, first unwind the "authors" subcollection.
- First filter for all papers from that satisfy the given filters (ex. year 2001 and venue "arXiv").
- Group by the element type provided. In this example: "authors", so we find the number of results for each author.

API:

1. Default request from navigation dropdown
   GET `/a5/trend3`
2. Request with user input parameters (sample request for above visualization)
   GET
`/a5/trend3?n=10&elementType=authors&filterKeys=venue&filterValues=ICSE`
3. Request for Query: Top years for papers starting with a ‘W’
   GET `/a5/trend3?n=5&elementType=year&filterKeys=title&filterValues=w.*`
   All parameters are required.

Trend 4: Collaboration of an Author

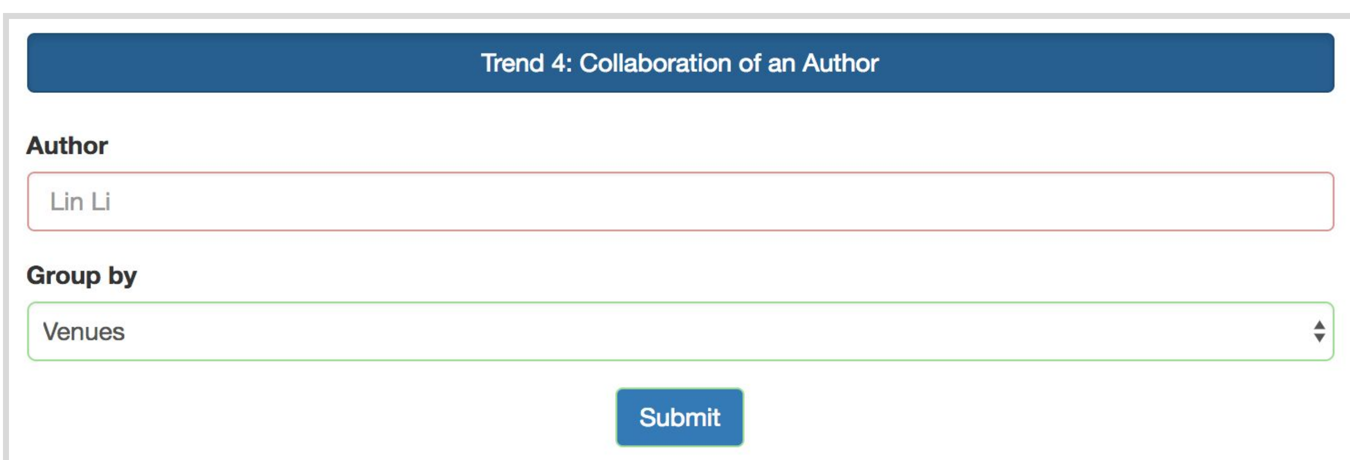

Figure 8.1: Query for Trend 4

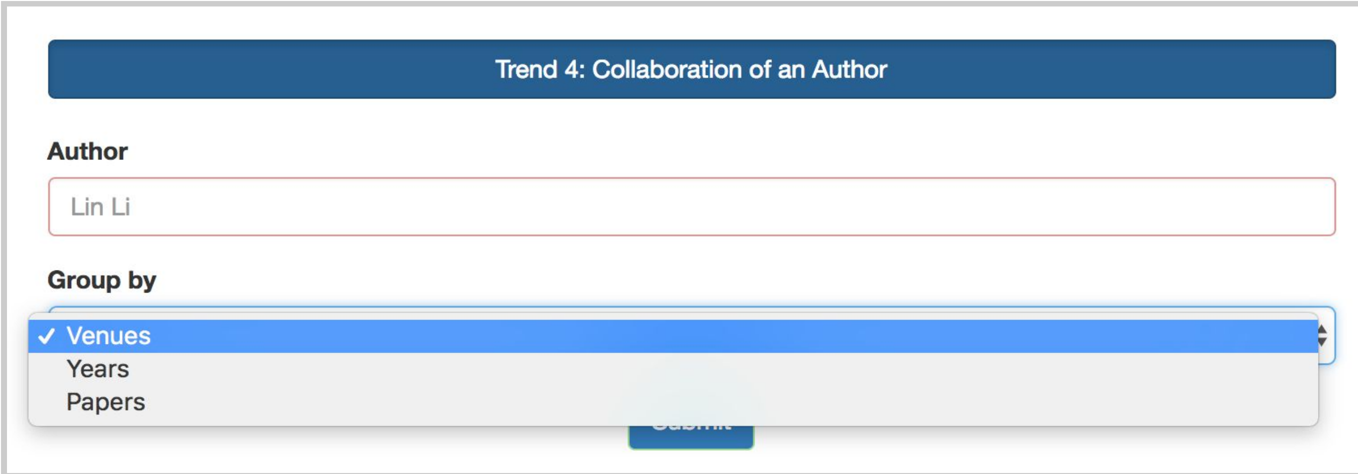

Figure 8.2: Dropdown for Trend 4

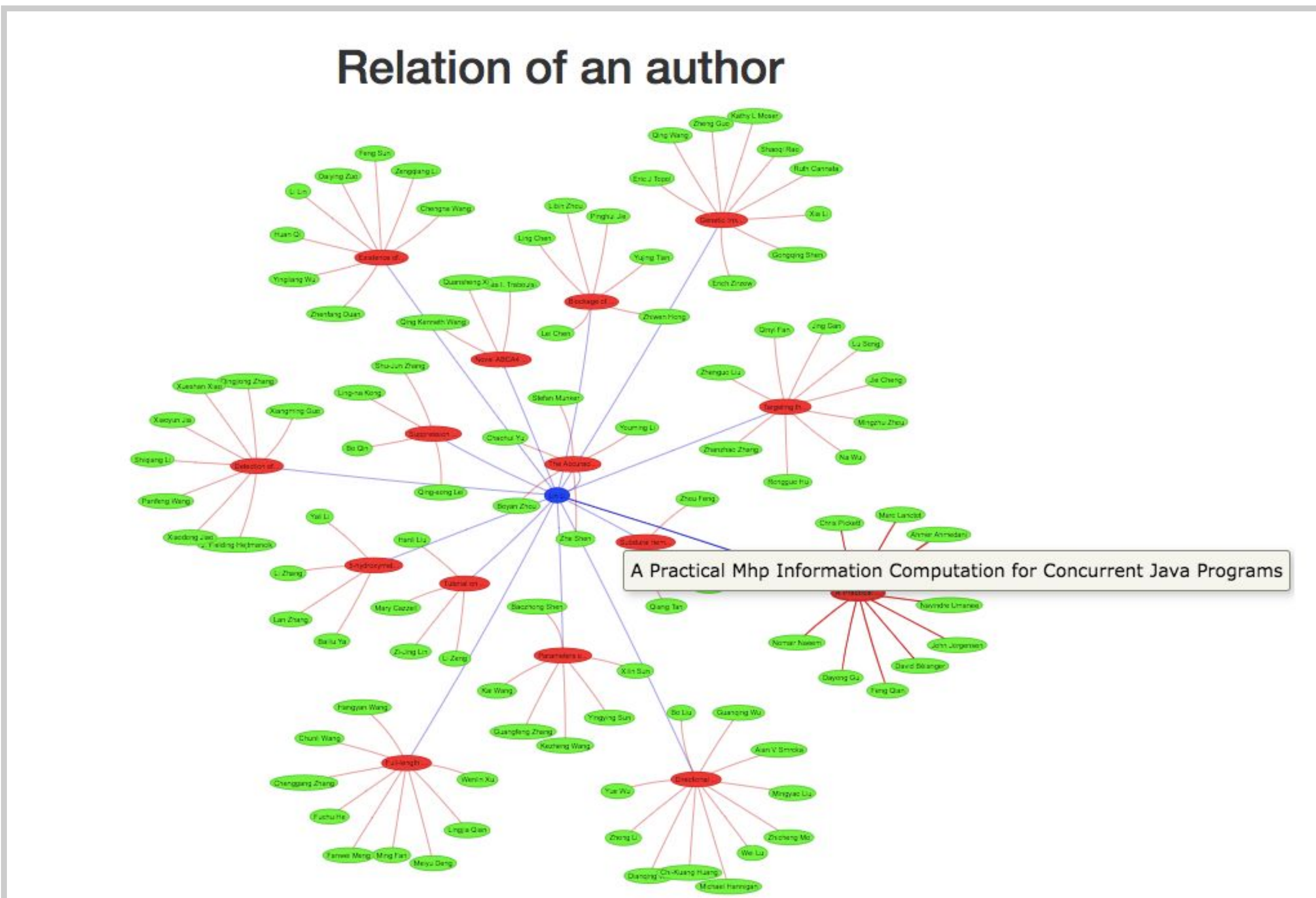

Figure 9: Graph Visualization

The purpose of this visualization is to see the involvement of a given author, grouped by either his papers, the venues or the year. For example, we can show all the paper with *Lin Li*'s contribution and and see the authors who helped him to do it. We can also group these authors per venue or per year.

The first argument is the name of the author we want to know the contribution and the second argument is the attribute used to group the authors together. In the figure above, *Lin Li* is represented by the blue node, the red nodes are the papers he contributed and the greens nodes are the authors who helped him.

Stepwise pipeline for generating this visualization:

- First filter all papers that contains the author we're interested in.
- Group all the authors by the attribute given (paper, venue or year)
- Create a graph by :
  - Creating a node for each group attribute
  - Creating a node for each author
  - Creating an edges between every authors and the group he is in

API:

1. Default request from navigation dropdown
   GET **/a5/trend4**
2. Request with user input parameters (sample request for above visualization)
   GET **/a5/trend4?author=Lin+Li&group=title**

   All parameters are required.

Trend 5: Citation Graph

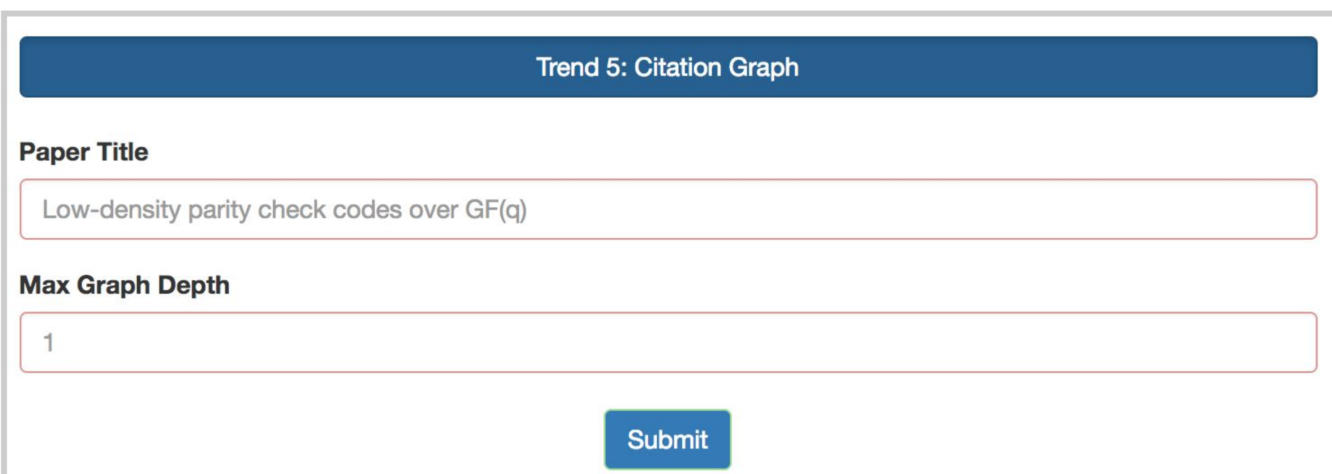

Figure 10: Query for Trend 5

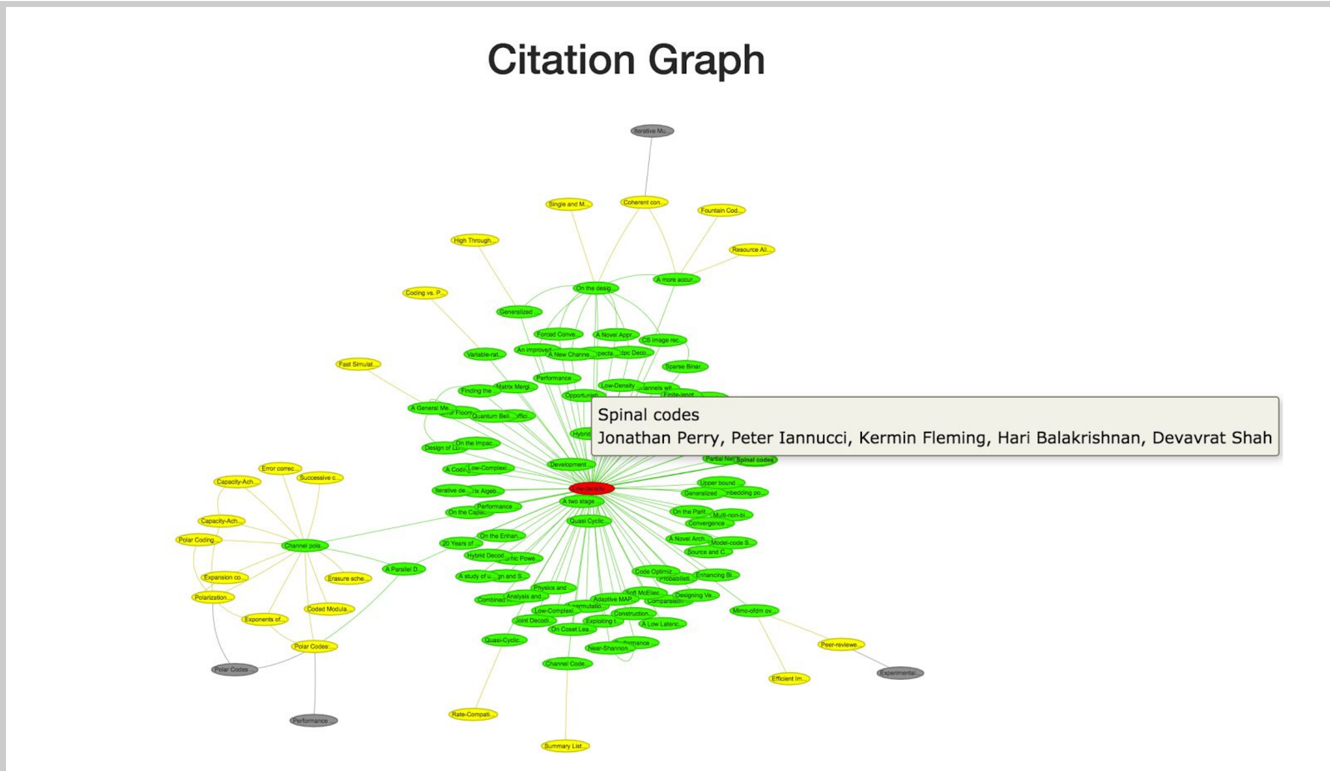

Figure 11: Graph Visualization

The purpose of this visualization is see how a particular paper is cited by other papers, up to a depth that can be provided as a parameter. In the graph above, the green items are papers that cite the root document. The yellow items are the second level, which cite the green items. The grey ones are level 3. One can hover over any node to see the full title and authors.

The parameters are the recursion depth (how many levels of citations to show), and the title of the root article (provided as a regular expression).

Stepwise pipeline for generating this visualization:

- First find the paper ID of the root node, for which the title regex was provided.
- Add the root node to the graph.
- For 0 <= i < maxDepth:
    - Find any documents that cite any node in the graph but are not yet in the graph.
    - Add these documents as new nodes with a new color to the graph.
- For each node, find the title of that paper and print that to the visualization instead of the paper ID.

API:

1. Default request from navigation dropdown

GET **/a5/trend5**

2. Request with user input parameters (sample request for above visualization)
   GET
   /a5/trend5?title=Low-density+parity+check+codes+over+GF%28q%29&maxDepth=3
   All parameters are required.