Academic Productivity Comparison Tool

by

Jason Andy Goble

Problem Report submitted to the
Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Bojan Cukic, Ph.D., Chair
Tim Menzies, Ph.D.
James Mooney, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2013

Keywords: Bibliometric Indexes, Data Mashups, Data Mining, Web Services

Abstract

Academic Productivity Comparison Tool

by

Jason Andy Goble
Master of Science in Computer Science

West Virginia University

Bojan Cukic, Ph.D., Chair

In the last decade, research output amongst universities has grown at a fast rate. Given the rise of technology and the easy accumulation of data, we can track and analyze the meta-information and statistics of sets of past papers. Organizations such as Microsoft and Google have aggregated the information for public use, turning these databases into a great resource for the professional researcher and for his or her peers. While the data is available to compare university productivity, the interfaces of these databases are lacking in producing a side by side comparison between universities. Using current web based technologies, we have developed an interface that facilitates the process of data comparisons between WVU and its peer institutions. This report describes the creation of a software tool that utilizes data found within Microsoft Academic Search to facilitate the comparisons across institutions.

Contents

List of Figures

List of Tables

Notation

We use the following notation throughout this problem report.

AJAX	:	Asynchronous JavaScript and XML
cURL	:	Client URL Request Library
MAS	:	Microsoft Academic Search
SOAP	:	Simple Object Access Protocol

Chapter 1

Introduction

The ability for a researcher or a faculty member to measure the research productivity of his or her institution is valuable. According to the increasing publication counts every year, university research is growing at an accelerated rate. Competition for funding and acquiring/improving academic reputation is fierce. With the wealth of research meta-information available combined with the automation of data mining, several projects have been created to aggregate publication data. Google Scholar, CiteSeer, and Microsoft Academic Search, have become monstrous databases of academic information. These projects have gathered millions of papers from journals, conferences and repositories and implement interfaces that allow users to search for authors, publication, and various other meta-data.

This problem report will focus solely on Microsoft Academic Search (MAS). MAS launched in December of 2009 and currently has over 38 million publications and over 19 million authors. The objects maintained are publication, author, conferences, journal, and organization. From Microsoft's interface, users can perform a free text search for any of the mentioned objects. For most academic researchers, their interest will mostly lie in the author and organization (university) data. The author profile provides a list of publications belonging to that author along with total citations of his or her body of work. The publication reference then links to a page that displays the abstract and links to the actual paper. Citations and references of the paper are also listed.

From a university standpoint, MAS is a great source of information. The database covers a multitude of academic domains and their subdomains. Microsoft keeps track of all of the publications associated with these domains for each university. Previously,

each author and university listed a computed h-index and g-index on the object page, but this has been removed, possibly due to the controversies surrounding the metric and the inconsistencies in data collected by Microsoft. Productivity Metrics will be discussed in the Related Work section.

Just like an author, a university has a set of publications that are associated with that institution. We can extract the citation counts and compute any bibliometric index we desire. This can also be performed on a yearly basis, and we can see how a university's research output rises or declines over time. MAS does not provide this information using its current interface. This problem report describes a series of exercises that attempt to provide visualizations displaying how productive universities are in relation to one another and over time.

Chapter 2

Related Work

2.1 Productivity Metrics

An obvious method to determine the research productivity of an individual is to sum the total citations of all his or her publications. Publication count can also be an indicator of productivity. However, a researcher can frequently produce low quality papers that are not cited by many other researchers. In 2005, Hirsch introduced "the index h, defined as the number of papers with citation number $\geq$ h, as a useful index to characterize the scientific output of a researcher." [5] Since then, the h-index has become the most popular way to measure quality of research. The h-index is not without its challenges though. Many researchers believe that the index may be too simple to model the complex nuances of problems such as accounting for high impact papers, self citing, or the age of the papers. The previously mentioned issues and their suggested fixes will be discussed later in this section.

As with authors, the h-index and it's various iterations can be used to measure research quality of any set of papers. This only holds true for papers withing the same field because, "There will be differences in typical h values in different fields..." [5] High publishing fields like the natural sciences will definitely have higher h-indexes than other fields. Given that the h-index of a university's entire set of papers combines different domains, this h-index may not be useful to compare two schools as a whole. However, we can assume that the average university has a majority of the domains, so as long as we do not specifically compare two different domains (Computer Science and Biology), the overall h-index of an university's papers is possibly a valid comparison. Though most of these bibliometric

indicators focus on the individual author, they can be used to evaluate the quality of a university's publications.

Most bibliometric indicators are calculated by generating a list of authors' (or universities') publications. This list is sorted in descending order by citation count (Table 2.1). To determine the h-index, in the table below we find the first paper N, whose citation count is less than its Document Number ($4 < 6$). The h-index is then the N-1 paper, in this case 5.

Document No.	Citations
1	16
2	10
3	7
4	6
⑤	6
6	4
7	2

Table 2.1: h-index example

The g-index was devised to factor citation count more heavily than the h-index. "the g-index is the (unique) largest number such that the top g articles received (together) at least g^2 citations." [4] To calculate the g-index, the Document Number is squared and the Citations are summed as we scan down. Here, we are looking for the Document Number whose Citation Sum is less than its square. In Table 2.2 the Citation Sum is less than it's square in row 8 ($51 < 64$). Therefore the g-index is 7.

Document No.(squared)	Citations(sum)
1	16
4	26
9	33
16	39
25	45
36	49
49	51
64	51

Table 2.2: g-index

Document No.(squared)	Citations(sum)
1	32
4	42
9	49
16	55
25	61
36	65
49	67
64	67
81	67

Table 2.3: g-index with highly cited paper

If an author has a single paper that is highly cited, it affects g-index but will not affect h-index. If we double the citations of this author's top paper from 16 to 32, his or her h-index will still remain 5. However, as seen in Table 2.3, the authors g-index will increase by 1, to 8. The h-index fails to account for authors who produce high impact work but not very many publications. The g-index works to correct this by factoring a cumulative citation count in the g-core of papers.

An improvement over h-index is that the g-index is not limited to the number of papers that a researcher has published. The h-index of a young researcher with 2 publications

can never be larger than 2, however, if these publications are cited enough the g-index can be much larger. Costas and Bordons [3] suggest that researchers can be categorized into four groups: High Producers that publish frequently but their set of papers has a lower citation count, Low Producers that publish infrequently and their set of papers has a lower citation count, Selective Researchers that publish infrequently but their set of papers have high average citations, and Top Researchers that publish frequently and their set of papers have high average citations. The h-index generally favors High Producers, while the g-index favors Selective Scientists. The effect of both indexes for Low Producers and Top Researches is less distinguishable. Costas and Bordons conclude that the h-index and g-index are complementary and neither should replace the other.

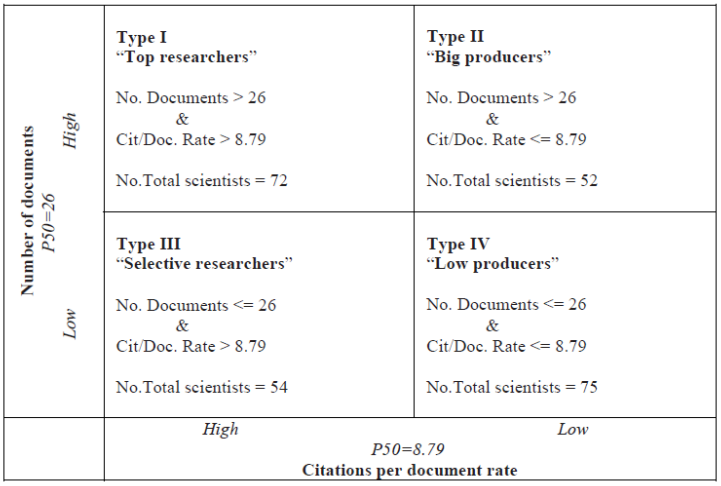

Figure 2.1: Categorization of researchers. Taken from [3]

Rather than use citation count like the h and g-index, ch-index tabulates unique citing authors per paper. "Ch-index is defined as the number such that, for a general group of scientific publications, ch publications are cited by at least ch different citers while the other publications are cited by no more than ch different citers." [1]

Document No.	Citations	Unique Authors
1	16	36
2	10	31
3	7	24
4	6	17
5	6	7
⑥	4	6
7	2	3

Table 2.4: The ch-index

The unique authors are larger than the citation count due to the fact that each citing paper usually has more than one author. The ch-index in Table 2.4 is 6, since the Document Number is greater than the Unique Authors ($3 < 7$). Document 1 actually has 43 authors out of the 16 papers that cited Document 1, but only 36 of these authors are unique. The ch-index does a good job at not weighing self citations or multiple citations from the author's peers. Further, someone who is cited by an abnormal number of papers that have a large number of unique authors, would have a skewed ch-index. The ch-index is more of a supplementary index to the h-index. A good utilization of the ch-index would be to compare the ch-index of authors with similar h-indexes.

The h-index does not account for the age of a publication. If one were to factor age into the h-index, influential works that were often cited in the 1970's should not hold the same weight as highly cited papers in the last decade. Sidiropoulos, Katsaros, Manolopoulos [10] suggest the contemporary h-index(hc-index), where a publication's citations are divided by their age (present year - publication year) and then multiplied by a coefficient lambda. Though lambda can change, it is most commonly set to 4. Citations of a paper published this year will count 4 times, while citations from a paper published 10 years ago will multiplied by a factor of 4/10. Publications are then placed in descending order of this aged citation count and the hc-index is calculated like an h-index with these differently weighted citation values. If we compute the contemporary index for the same data as above:

Document No.	Citations(age factored)	Publish Year
1	8	2004
2	4.8	2007
③	4.7	2006
4	3.6	2001
5	2.7	2006
6	2	2004
7	1.1	2005

Table 2.5: Contemporary Index

The hc-index in Table 2.5 is 3 because the adjusted citations are less than the 4th Document Number ($3.6 < 4$). In general, hc-indexes will be smaller than their h-index counterparts. The creators of this index also suggest weighing the individual citations of a paper by the citation's year. Groundbreaking papers that are still relevant today are not accounted for using this contemporary index. If a paper is published in 1970 but still receives citations in recent history, objectively, it should not be penalized by a factor of 32. Sidiropoulos, Katsaros, and Manolopoulos also suggest that a modification of the index can be used to account for this problem. Instead of looking at the publication year, citations are weighed individually by the year they occurred. A citation in 1970 will be weighed much less than a citation that occurred in present day for the same publication.

Hirsch himself has come up with a new variation of his h-index and proposed "the index hbar, defined as the number of papers of an individual that have citation count larger than or equal to the hbar of all coauthors of each paper, as a useful index to characterize the scientific output of a researcher that takes into account the effect of multiple coauthorship." [6]

The hbar assumes, that papers which have co-authors with higher h-indices generally receive more citations due to the influence of this senior author. So, if a co-author has an h-index greater than or equal to the citation count of that paper and greater than or equal to the h-index of the main author that paper will not be counted in the h-core papers of the main author. For instance, a young researcher author A with h-index of 10 has coauthored a publication with a veteran researcher B who has a h-index of 42. This publication has

30 citations, therefore it would not be considered for the hbar calculation.

2.2 Academic Databases

In this section a brief overview of other academic databases is provided and how they compare to MAS. Each project maintains their own database and publications and the level of accuracy of data will vary in each. Many of the projects are secretive of their algorithms for gathering data, so it can be difficult to determine which one has the most complete content or quality content. Most of the content is gathered via web-crawlers and scraping of HTML, PDF, and Word documents. Existing databases are also scanned for relative information. However, the automation can lead to duplicate papers and incorrectly assigning papers to authors. Nie, Ma, Shi, Wen, Ma [9] of Microsoft relate building an academic database to a web search engine. Information is first extracted from web objects like HTML and PDFs. The information that achieves a higher ranking is more likely to represent the Author or Publication as it is in reality. These researchers worked on Libra, a Computer Science publication search engine and database. Methodologies used in Libra have been applied to MAS. MAS documentation itself admits that it is built upon data from projects that are older than itself such as DLBP and CiteSeer.

Digital Bibliography and Library Project (DLBP) is a database that has been growing since the 1980's and contains publications in the domain of Computer Science. DLBP provides no API but does provide downloadable XML for data mining. CiteSeer focuses on Computer and Information Science as well. Over the past decade, it has been decommissioned and then revamped as CiteSeerX and now has over 20 million publications. Since these databases mostly focus on one domain, they do not meet our goal of comparing university data at the top overall level.

Google Scholar is the most similar to MAS in its scope by covering many academic domains. A programmatic way to reach Scholar data is not yet available to the Google AJAX API. Scraping the information via web bots from Google Scholar is not allowed according to their End User License Agreement and IP's that try will be blocked. Publish or Perish is a software tool set and companion book that analyzes Google Scholar data, particularly citations. It can be inferred that it scrapes the data from Scholar in some

manner that does not block the software. This application uses some of variations of the h-index previously discussed such as the contemporary index. Publish or Perish focuses on authors and not universities. Information on Publish or Perish can be found at `http://www.harzing.com/pop.htm`.

The consensus varies from source to source and person to person when comparing Google Scholar and Microsoft Academic Search. Some users claim MAS to be the most accurate and others vice versa. It can be stated that MAS is probably more precise in some domains while Scholar is in others. When quickly comparing Dr. Bojan Cukic in both databases we get the following: Dr. Cukic has 254 articles in Google Scholar with a h-index of 23 versus 154 publications in MAS with a h-index of 17. Most Authors will be more satisfied with the database that shows the most accurate portrayal of their work. A higher h-index on record for all to see is also more desirable. Scholar has been in existence longer than MAS, so it can be argued that Scholar is more complete in terms of data due to time.

MAS already provides a plethora of information and several visualizations. A number of Author relationship tools that are rendered using Microsoft Silverlight have been created. These can be found at: `http://academic.research.microsoft.com/VisualExplorer#525735`. For instance, using MAS data, you can start with an author and view the tree of collaborators and other authors that cite the parent author. Microsoft also allows users to compare the publication and citation output of two universities. However it only allows this visualization for domains. This visualization can be found at: `http://academic.research.microsoft.com/Comparison?entitytype=7&id1=552&id2=405&topDomainID=2&subDomainID=0`. There is a need to compare universites at a top level without drilling down into domains. The University Comparison Tool fills this need.

2.3 Limitations of Microsoft Academic Search Data

Most faculty members start their careers at a university different from their final tenure destination. The way MAS works, all papers by an author will belong to all organizations that he or she has been affiliated with throughout his or her career. If an author writes a paper at the beginning of their career at University 1, and later on moves to two different universities, this paper will be associated with all three universities. When attempting to

compare university productivity, ideally we would like to have the most accurate data and logically a publication that was written under the roof of another university should not be factored into another university's publication count.

If an author has a common name, he or she may be lumped with other authors that share the same name. This could cause inconsistencies in a university's publication and citation count. If the lumped author is associated with university A, those invalid publications that did not belong to the author would be effectively "stolen" from other universities.

Chapter 3

A Tool For Comparing Research Productivity

3.1 Microsoft Academic Search API

There are several potential solutions to the proposed problem. Using scripting languages such as PHP or Python, a programmer can impersonate a web browser and make requests to the MAS site. For example, since the site utilizes GET parameters, we could formulate the following url query `http://academic.research.microsoft.com/Author/525735`. This would return a web page with information regarding an author. We could then parse the html and follow the links to his or her publications. This approach has several problems. While making the request is rather trivial, automating the proper parameters can prove difficult. There are many libraries available to assist in the extraction of information from html. However, even if the pages belong to the same site, the structure of the html for MAS pages could vary drastically among different object types. Developing code to parse each specific html page would be tedious and inefficient. Building a database from scraped files, would most likely take an extended computation time. Every time a user would want up to date information, the scraping code would have to be run again.

Fortunately, Microsoft has provided an Application Programming Interface (API) with their project. This is a web service that allows anyone to programmatically query data from MAS. While not a true interface to their database or a guide to understanding the inner workings of the project, users of the API can still have access to a wealth of data. The API allows access to seven objects: Publication, Author, Conference, Journal, Organization,

Domain, and Keyword. For the purposes of this problem report we will mainly be dealing with Publications and Author queries. The API provides data from 1960 to 2013. As we get closer to the present, the data becomes less complete. It can take a couple years for a paper to find its way into the MAS database and for citations to accumulate. Therefore we will only use data between 1960 and 2010.

Queries in the API are not made like typical database queries. When trying to retrieve an object of any type, there is a set list of parameters used to specify the required data. These parameters are similar to the comparison predicate of a WHERE clause in SQL. However, the query parameters may not always be relevant to the selected object. For instance, nothing is returned when you provide an AuthorID when querying for an Organization. If you were to reverse the query, the API will return all Authors that correspond to the UniversityID provided. Although the API has decent documentation, it takes some trial and error to discover what queries are actually possible.

The Microsoft Academic Search API allows users to access their data in one of two ways. One can get results via SOAP (Simple Object Access Protocol) or via JSON (JavaScript Object Notation). SOAP is useful for static languages such as Java and C. However, SOAP can often increase complexity of a project. The only benefit to SOAP mentioned in the API's documentation, is that requests do not have a maximum parameter limit like JSON. Experiments in the current project have not reached the JSON request parameter limit. When building requests for SOAP, each request is built like a strongly typed object. Parameters are added as attributes to the request object. When building request objects for JSON they are basically just creating urls for a HTTP get request.

```
http://academic.research.microsoft.com/json.svc/search?AppId=
AppID&ResultObjects=Publication&OrganizationID=552&StartIdx=1&EndIdx=
10&OrderBy=CitationCount
```

Figure 3.1: URL of GET Request for a Publication object

AppID is the application ID provided by Microsoft after sending a request. This query returns a list of the top ten most highly cited papers that belong to any professor that is or has been associated WVU (552 = WVU).

3.1.1 Limitations of the API

When retrieving a MAS object all of the data corresponding to that object is returned. Unlike SQL, specific columns of the data you desire cannot be specifically queried for. Many times you will only require one attribute of the data. The information that you do not need only makes the request slower. Another factor is speed of the queries. Each query can only return 100 objects at a time. When querying for a Publication object, which is the largest given the meta-data, on average it can take 0.25-0.5 seconds. Querying for 100 Publication objects, takes around 1 second. For example, we wish to calculate the h-index for Harvard since 1960 across all domains. The API does not return h-indexes for universities, only authors. We must query for all Publications associated with Harvard until we find the h-index. The Harvard overall h-index is 635. So we must make at least 7 queries each returning 100 Publications. So the query response time alone, not including number calculations would take approximately 7 seconds. Part of the application calculates the h-index a number of times N, where N is the number of years. So the worst case scenario (Harvard has the largest h-index of schools) is 50 * 7 seconds. These request times can be mitigated by making concurrent requests, which will be discussed in the Design section.

SELECT COUNT(*), SUM(Citation_Count) FROM Publications WHERE University_ID = 512 and Year $\leq$ 2000 and Year $\geq$ 2010

Figure 3.2: Possible SQL query for MAS

```
http://academic.research.microsoft.com/json.svc/search?AppId=
AppID&ResultObjects=Organization&OrganizationID=552&StartIdx=1&EndIdx=
1&YearStart=2000&YearEnd=2010
```

Figure 3.3: Actual MAS query

3.1.2 API Errors

As with any web service, reliability is not perfect. Given the number of queries that are made in the application, some will eventually fail. Errors currently fail silently. There are also certain queries that will consistently fail or produce bad data for an unknown reason.

When trying to access Computer Science Publications , the API will return irregular data for years 2003 and 2009. The data is unrealistically high regardless of what university is selected. Issues such as these are almost certainly small problems on Microsoft's end and they will need to be contacted for any resolution to occur.

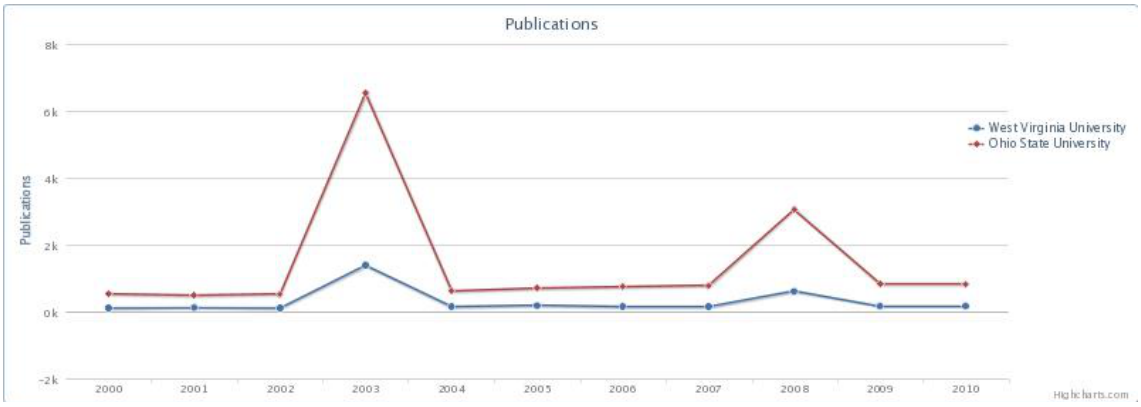

Figure 3.4: Disproportionately Large Data for 2003 and 2009

3.2 Design and Technologies Used

Creating the project as a web application allows users to access the service with any web browser (JavaScript enabled) and also allows for concurrent users. We wanted to provide the user with an interactive GUI, so JavaScript with AJAX requests were used. JavaScript allows us to utilize interactive event-driven display and libraries such as JQuery and Highcharts. JQuery is a cross-browser compliant wrapper built on top of regular JavaScript that reduces the code written and enables us to access the HTML DOM with additional ease. AJAX (Asynchronous JavaScript And XML) is a combination of technologies that can create partial HTTP requests that only updates part of the page, providing a desktop application feel to the web browser. Though misleading, AJAX is moving away from XML transitioning to JSON for data serialization which will be described below.

With JavaScript/XHTML on the client side, the server-side language needed to work efficiently with those technologies. Dynamic scripting languages such as PHP and Python are better suited for smaller projects. Static languages such as Java or a language in the .NET framework tend to complicate small projects with over-engineering architectures. PHP was chosen due to its ease of use with web services and overall library support.

MySQL was used as the database for the small amount of data that needs to be persisted.

Given that we chose PHP as the server-side language, we chose JSON for its lightweight interaction with the language. During a HTTP request, data cannot be represented in the object state that it was on the server. The data must be serialized in some fashion, and then re-encoded on the client side. JavaScript has implemented libraries that re-encode the data into an object representation that is understood by JavaScript. In the overall architecture of a request, JSON is actually used twice. First, the user utilizes the JavaScript interface to send an AJAX request the the PHP code. Then PHP, queries the Microsoft Academic Search API. This data is received as JSON from Microsoft and is then decoded into a PHP object and then processed. The processed data is then decoded to JSON as a reply to the original request. JavaScript encodes the JSON string into a JavaScript object and the data is passed to the graphing module.

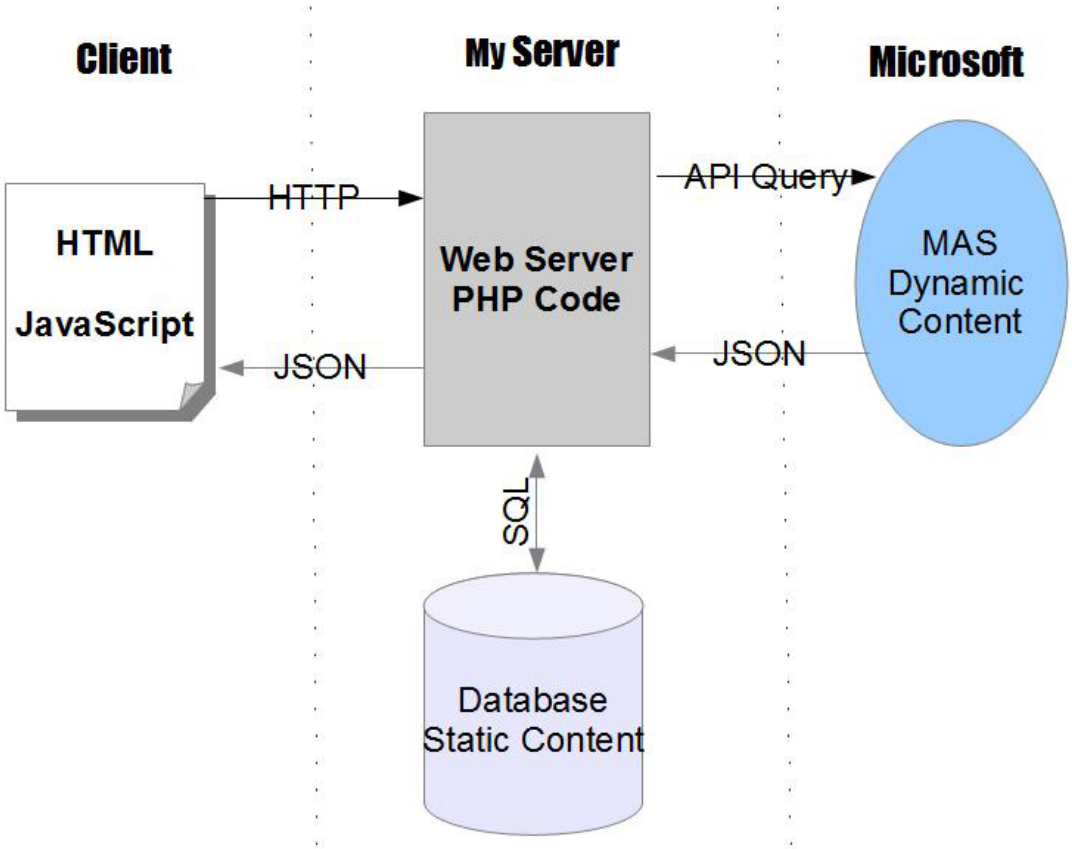

Figure 3.5: Architecture of Comparison Tool

3.3 Academic Productivity Comparison

The tool can be found at: `http://boughtnsold.org/fastMAS/MAS.html`

The academic comparison tool is the most complicated portion of the code written. While the majority of the data displayed by the tool is gathered in real time from the API, it is practical to mirror some of the more static data. Data such as a list of universities and a list of domains were collected. This was performed by writing PHP scripts that grabbed all universities in MAS that were associated with the full text search "Computer Science". The university name and MAS ID for each school were inserted into a MySQL table. There are approximately 7000 universities in the university table. The extraction process was performed for 12 domains and 205 subdomains.

To allow users to order universities by overall h-index or g-index, it was necessary to calculate these values for all universities and store them in the database. While you can order universities by citations etc. via the API, there are no indexes available. Calculating indexes on the fly would be nearly impossible. The largest g-index is over 1000. This means, for each school in the database, 10 API queries would need to be performed. Both indexes can be calculated with the same data, however the process takes a long time with 7000 universities. The h-index and g-index will both increase slowly over time, so in order to maintain accurate numbers this scripting process needs to be run periodically.

The interface allows users to type in any popular university by name, and the field will auto complete the closest 10 universities that match. An AJAX request is fired for each character that is entered after the first two and performs a wild card query on the string against the MySQL database. Once the desired school is selected, it is placed in the compare area on the web page. Users can also press the Select School button, which will produce an additional window that allows users to search for universities. They can search a range of schools ranked by both g and h-indexes or search for similar indexes. The selected schools will be added to compare area. Once, the desired schools are selected, the user must press the Compare Schools button.

JavaScript sends an AJAX request to the main PHP controller. This controller is a PHP script file that delegates the various MAS API queries that need to be made. The data sent from the interface is the list of school ID's and the year range, plus any domain or subdomain. It should be noted that the application can only compare ten schools at a time.

For each school and each year in the range the application will have to make at least two queries to the API. So, if we calculate the entire 50 year range (1960 to 2010) for 10 schools we will have to make 1000 queries. This is the worst case scenario for the application. The main data that fuels the application is citation count and publication count. These values are received from the API using the Organization object. The controller launches the code to query for the publication and citation count for each year. The data points are then added to a PHP array that is then added to an object representing a point of data to be shipped back to the client side via JSON. The calculations for the normalizations are performed on the client side using JavaScript.

3.3.1 Normalizations

Certain universities obviously have resource advantages over other universities. It is a priority of this application to experiment with the various data available in MAS to present a clearer picture of research productivity. By normalizing the publication and citation counts of each individual year, we hope to show a view school productivity that is not normally visible to the user.

The MAS API allows us to query the Author object and return the number of unique authors that published for a specific year, in a specific domain and are associated with a university. For example, if a query is made that asks how many active authors in WVU's Computer Science subdomain of Software Engineering there were in 2008, the resulting count is 13. We can then manually query for all publications using the above criteria, then double check the list of authors and ensure that they appear on the publication list. The author counts are added to an array and sent to the client side. The user can then select the Author checkbox then click the Normalize button. The graph will then show publications or citations per author for each year.

$$\frac{P_i}{A_i} \quad i \in \{y_1, ..., y_N\} \tag{3.1}$$

$$\frac{C_i}{A_i} \quad i \in \{y_1, ..., y_N\} \tag{3.2}$$

The data can also be normalized by the h-index. Since the h-index can be an indicator of the quality and impact of the work, multiplying the publication or citation count by a h-index coefficient of the set of papers for a specific year. Calculating the h-index for each

set of papers for each year can be expensive in terms of time. The largest h-index for any school per year is less than 300, so we must query at least 300 publications for each year, and we can have 50 years for each school. There are two ways of handling the h-index calculation. We can make 3 sequential queries in a loop and if we find the h-index before the last query we can break from the loop to avoid unnecessary work. The requests can also be sent simultaneously, but will have to make 3 requests for each year, even for schools that may have an h-index less than 100. After writing code to test the speed of both methods, sending concurrent request is generally ten times faster even though we ultimately end up making more requests. Though concurrency is the best solution it poses some issues. The API has a request per time limit that is easily reached when the requests are made so frequently when concurrent. This action will lock our API account temporarily and break the application for that time. More about this issue will be discussed in the Optimizing API calls section. However, when we calculate the h-index for the purpose of normalization, we are only making at most (50 x 3 x number of schools) queries, which should not put us over the limit.

Each point of data whether it be, publications or citations, is multiplied by 100th of the h-index for that year.

$$\frac{P_i h_i}{100} \quad i \in \{y_1, ..., y_N\} \tag{3.3}$$

$$\frac{C_i h_i}{100} \quad i \in \{y_1, ..., y_N\} \tag{3.4}$$

University reported federal research funding was taken from The Center for Measuring University Performance. This data for over 600 universities is in current US dollars and ranges from from 1990 to 2008. The data was exported to csv and then the university names in the file were matched to the current university names in the MySQL database. The funding amounts and corresponding year are stored in a separate table that references each university by its MAS ID. The funding data is pulled from the database and sent back to the client in a JSON object. Publications or citation counts are divided by the amount of funding in millions of dollars. So, the more research funding (resources) a school receives the more of an advantage that school has in generating more publications per year. If a user tries to normalize the data by funding, but selects a year range outside of 1990 to 2008 the application will not graph the data. The data outside of the range would ruin the

graph.

$$\frac{1000P_i}{F_i} \quad i \in \{y_{1990}, ..., y_{2008}\} \tag{3.5}$$

$$\frac{1000C_i}{F_i} \quad i \in \{y_{1990}, ..., y_{2008}\} \tag{3.6}$$

The application allows the user to normalize the data by all three coefficients or any combination of the three.

$$\frac{10P_ih_i}{A_iF_i} \quad i \in \{y_1, ..., y_N\} \tag{3.7}$$

$$\frac{10C_ih_i}{A_iF_i} \quad i \in \{y_1, ..., y_N\}$$

(3.8)

However, since each coefficient is of a different magnitude, the results are highly skewed by the larger factors. It is better to view each individual metric by itself.

3.3.2 Optimizing API calls

Initially during the first implementation, the data for the Comparison Tool was gathered sequentially. For each university selected and each year in the range, an Organization query, an Author query, and one to three Publication queries were made. Each of these queries could take up to 1 second to complete. Therefore, when there were large requests, say four universities for a period of 30 years the application could take an unacceptably long time.

A PHP library called cURL, commonly used to send HTTP requests and automate web-crawling, has functions that allows the user to send a large number of HTTP requests at once. Code was written to test the library and the results were excellent in terms of speed. The cURL library could easily ship over 100 requests to the Microsoft Academic Search API and Microsoft would send the results back in several seconds. However, it was discovered that the API only allows 200 requests per minute to prevent clients of the API from overusing Microsoft's system resources. If this limit is broken, the API will cease returning data and require human interaction or will reset automatically after a period of time. The human interaction is a CAPTCHA-like question that requires an actual user to add two numbers. If the requests the University Comparison Tool makes break the limit,

the application will not be able to show the results from the queries generated by the user interface.

In testing concurrent requests, the limit was actually less than 200 per min. Through further experimentation, 90 individual transactions per approximately 30 seconds is the most that could be consistently made with locking the API. For the Comparison Tool, for each data point on the graph we must do an Organization, Author, and 3 Publication queries (if we require a h-index calculation). Therefore, for each data point we require 5 single queries to the API and only (90/5) 18 points of data per 30 seconds. However, in practice sending a large number of requests was returning null data for some points. This was probably due to the larger size of Publication requests. Note that when querying for Publications we are requesting 300 actual publications for their citation counts. When querying for an Organization or Author we are just asking for one result, because the data we require from MAS is included in the meta data. So, naturally Publication queries require more time due to the quantity of data. To combat this delay, the user interface allows a checkbox that turns h-index calculation on and off and reduce the number and size of the queries. If we do not request any Publication objects, then we can calculate 45 points per 30 seconds. When querying for Publications, we reduce the batch size down to 50 requests (10 data points) per 45 seconds. This setting tends to prevent any null data.

Comparing two universities from 1990 to 2010 (with h-index calculation) and using the old method of sequential requests, it takes a time of 8 minutes and 30 seconds. Requesting the same data using batches of concurrent requests, lowers the processing time to approximately 5 minutes.

As with any implementation involving concurrency, additional complexity is added. cURL performs concurrent requests by placing them in an array. A corresponding array is returned, each element a JSON response from MAS. Compared to the sequential way of making requests within a loop, implementing concurrency is more difficult. In each iteration of the loop we know which university and year the point of data belongs to.

```
foreach (university as u)
{
    for (i=startYear; i<=endYear; i++)
    {
        Organization.makeQuery(u,i)
        Author.makeQuery(u,i)

        //may only get 100 publications, 300 max
        hIdx = computeHidx(u)

        push data onto university array
    }

}
```

Figure 3.6: Sequential way of making requests

Sending approximately 80-90 requests at once requires proper tracking once the results are returned to ensure that they get placed in the proper array in the right order. To ensure speed, we want to place as many requests as possible in each batch. Therefore we do not have full control over the ordering of the requests/responses. The first result in the batch could be WVU for year 1993, while the last could be another university for year 2003.

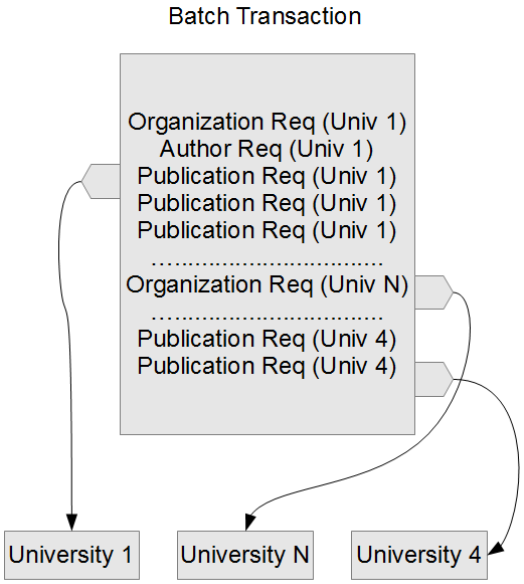

Figure 3.7: A batch of requests/responses and ensuring they match the correct university

3.4 Domain Popularity Analysis and Prediction

The tool can be found at: `http://boughtnsold.org/fastMAS/domain.html`

Within each domain of university research, it is practical to know which subdomains of

a field are going to grow in the future and which domains will stagnate or dwindle. University faculty may be concerned with choosing an emphasis that differs from the trending subdomain or may choose the popular path. Using a part of the GUI code written for the University Productivity Comparison Tool, another tool was created to trend domain data. A user starts by choosing the top domain, then they choose the two subdomains to compare with each other. Comparing two subdomains that are not related, such as Data Mining and Astrophysics is not as meaningful as comparing Data Mining and Natural Language Processing.

The data depicted on the graph by default will be the publication count in each subdomain across all MAS Organizations for each year selected. If the Cumulative checkbox becomes selected, starting from the second year, the y-value will contain the total of all publications of the previous years on the graph. Note that we do not include an initial offset from the years prior the start year. For instance, if we decide to graph data from 1990 to 2010, the initial data will start at 4683 and will not include the 470 thousand publications from 1960 to 1989.

For each of these graphs, the geometric mean rate of growth is calculated and displayed. This is the average percentage at which the publication count must grow each year from the starting year to the end year, to reach the exact value of the end year.

$$(\frac{End\_Publications}{Start\_Publications})^{\frac{1}{Years}} - 1 \tag{3.9}$$

We can approximately fit an interpolated line using the scatter plot of the publication counts [8]. Using simple linear regression we plot a line using time as the independent variable. Linear regression has been used to predict and trend data. Given a range of years we perform linear regression on that data and using the slope and y-intercept we graph 10 years into the future. For instance, if the range 1990 to 2010 is provided, the line will be extended to the year 2020. The regression of the line was created from the 30 years, from 1990 to 2010. The line from 2011 to 2020 may shed some insight on the future values of publication count in the coming years. For the cumulative publications, the line will never have a negative slope due to the publication count always growing. For the publications per individual year, the line could possibly have a negative slope and a negative mean rate of growth. Given that most subdomains have had an exponential growth in publications

in the last 30 years, using data before 1990 will lower the predicted values.

3.5 Sanitizing Author Publications

The tool can be found at: `http://boughtnsold.org/fastMAS/PDFsearch.html`

As mentioned, given MAS's automated data collection and organization, information can be incorrect. Authors with common names can often be lumped together and an arbitrary author from the group can be erroneously credited with work. To see if we can try to automatically detect publications that do no belong to a particular author, we have created a tool to search through an authors papers and flag possible papers that meet a certain criteria. The user starts by providing a MAS user ID that can found by searching the Microsoft site. It is best to have the user find the particular researcher from there to ensure that the author name matches the desired university and the authors papers can be viewed. The ID can be found in the format `http://academic.research.microsoft.com/Author/XXXXX`, where the X's compose the user ID.

The user ID is then applied to a query to the API which gathers a list of the author's publications. Within the return results is a link to the location of the publications at various repositories. For the purposes of this exercise we are only interested in publications that are found in the IEEE database. Publications that have IEEE entries are selected by a unique string that is found in IEEE URLs. A unique identifying number is extracted from the string and a new URL is created to access the publication from WVU library's IEEE access.

From outside WVU's network, all library databases are behind a user name and password. To achieve this login programmatically, the cURL library was again used. The cURL library allows us to impersonate a user agent, such as Mozilla Firefox. We provide the code with my personal WVU user name and password and cURL sends the credentials to the university library via POST request. Once logged in the code saves the unique login token from the university web server in a cookies.txt file for future login attempts. A request is then made using the URL that was built to locate the PDF file of the publication. While in an actual web browser the PDF will display, there is one more step needed to automatically scrape the file from the IEEE database. The file is embedded in an iframe element

and another link must be extracted to reach the actual PDF data. A request is sent using this link and the PDF is loaded into application memory. The PDF is encoded in Adobes proprietary format and we cannot search for keywords while the data is in this state. There are no libraries in PHP to decode the PDF into a computer readable string. Instead we save the encoded data to a temporary PDF file. The tool pdftotext, from the open source project XPDF, is called from the PHP code and returns a string of the PDF content. The data is searched for all of the keywords and if one is found , the document is marked as valid.

Using the external tool is not optimal, due to the fact that we must perform two I/O operations for each PDF file that MAS has assigned to the author. PDF files are only scraped for IEEE documents, but the code could be modified to search any repository. WVU provides access to all major academic repositories, so the login code would still be valid. Each individual site would require specific html scraping code to ensure that the software would find a way to the PDF.

A WVU faculty member named David Graham (Electrical Engineering) is a good example to demonstrate the effectiveness of the sanitization. `http://academic.research.microsoft.com/Author/17992829/david-w-graham` Due to his common name his work has been merged with other academics that share the same name. When we enter his MAS ID into the application it will scan all IEEE PDF's referenced in MAS. Of the 42 publication in the IEEE database only 10 contain information concerning WVU. If we look at the 32 that were marked as invalid they are either in another domain and do not belong to David Graham or actually belong to David Graham but were from a previous university positions. The introduction of natural language processing to the PDF search could further improve the sanitization. For instance, using the keywords associated with a MAS author and searching for synonyms of these words in the PDF would filter out invalid publications in the wrong domains but keep publications of the correct author when he or she was at different universities.

3.6 A Sample Comparison

In this section, a comparison will be performed between WVU and its peer institution Virginia Tech.

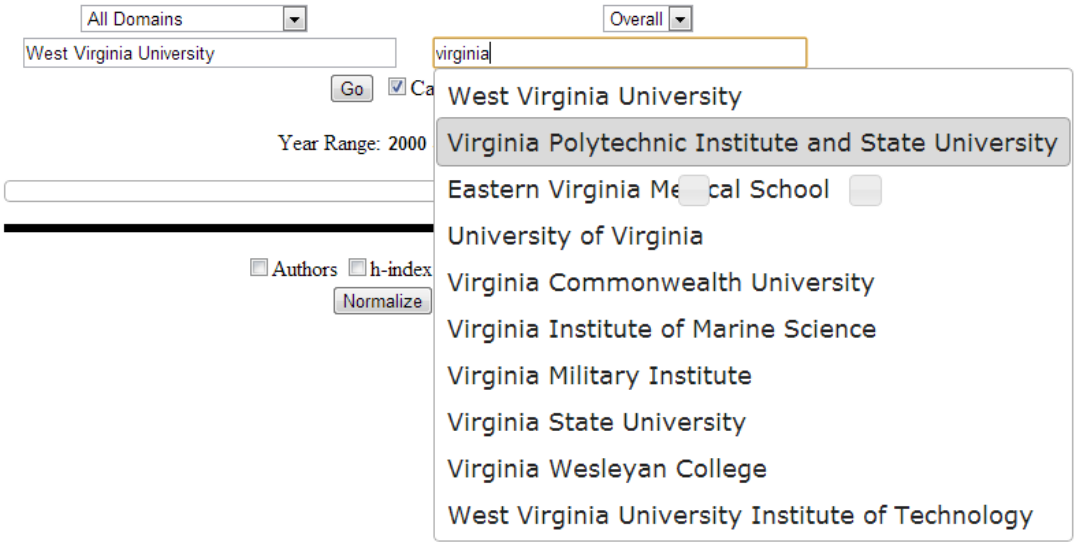

Figure 3.8: Select the universities

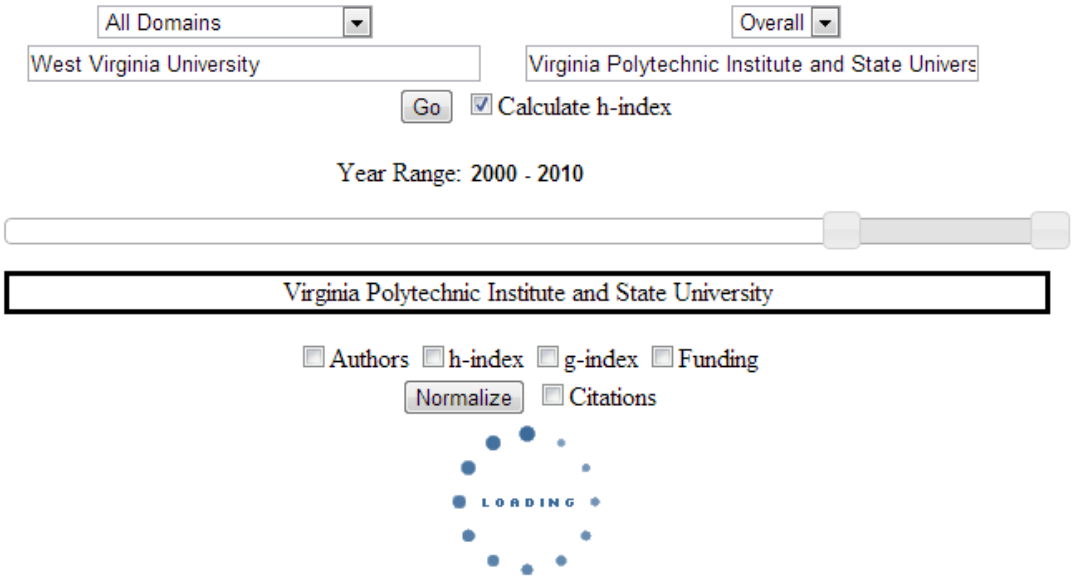

Figure 3.9: Press Go

All universities in the black bordered rectangle will be compared with WVU. Nine other

universities can be added. However, this will increase request time. Note that the calculate h-index checkbox is selected, therefore 3 Publication queries per data point will be made.

You can also select a specific domain and subdomain. For this example, the tool is grabbing publications and citations in all domains. Once the data has been retrieved from the API, the user can check any of the normalization boxes. Once a box is checked, click the Normalize button to change the view of the data. If the Citations checkbox is selected, Publications will be replaced by Citation count for that year.

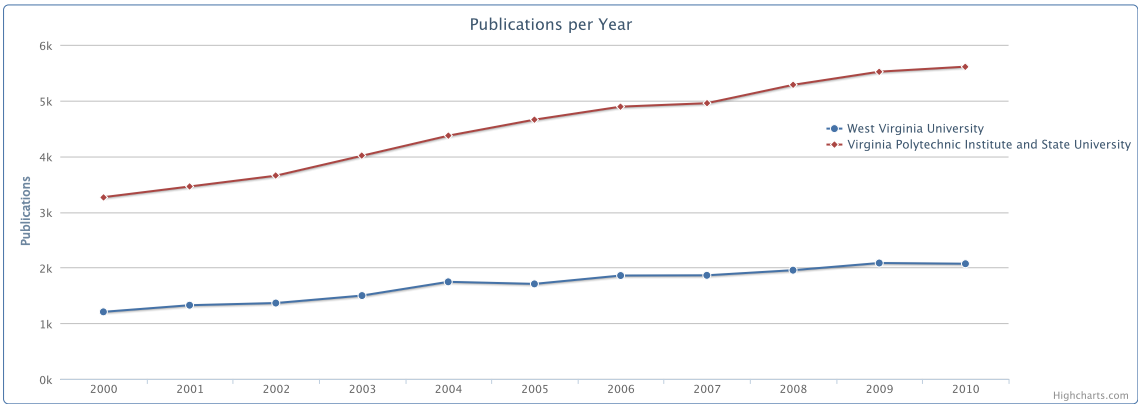

Figure 3.10: Publications per Year

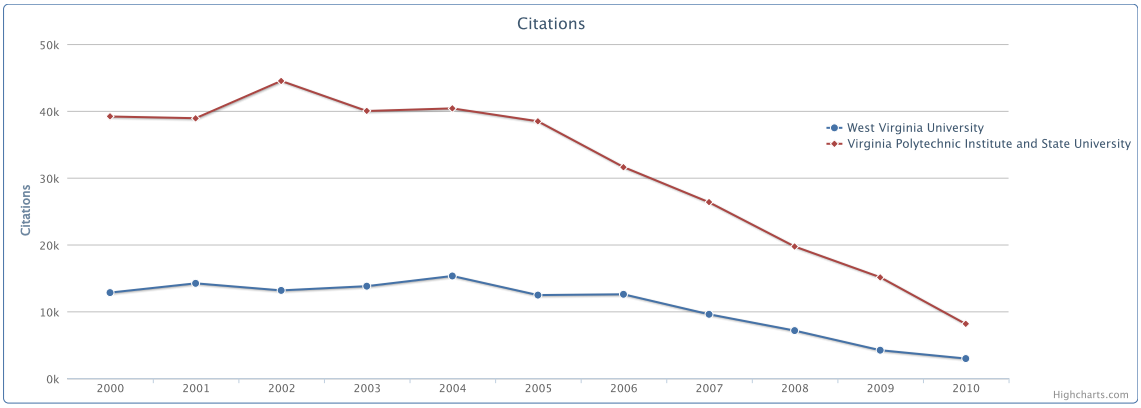

Figure 3.11: Citations per Year

One can notice how citation counts taper off towards the present. This is a result of the fact that citations for these publications are still being accumulated.

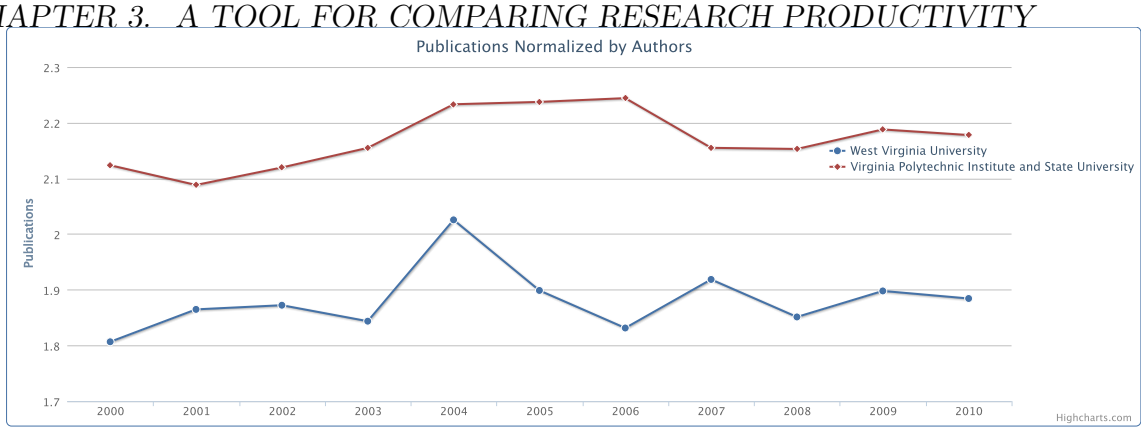

Figure 3.12: Publications per Author

In 2004 WVU has a spike in Publications per Author that brings its level closer to Virginia Tech's.

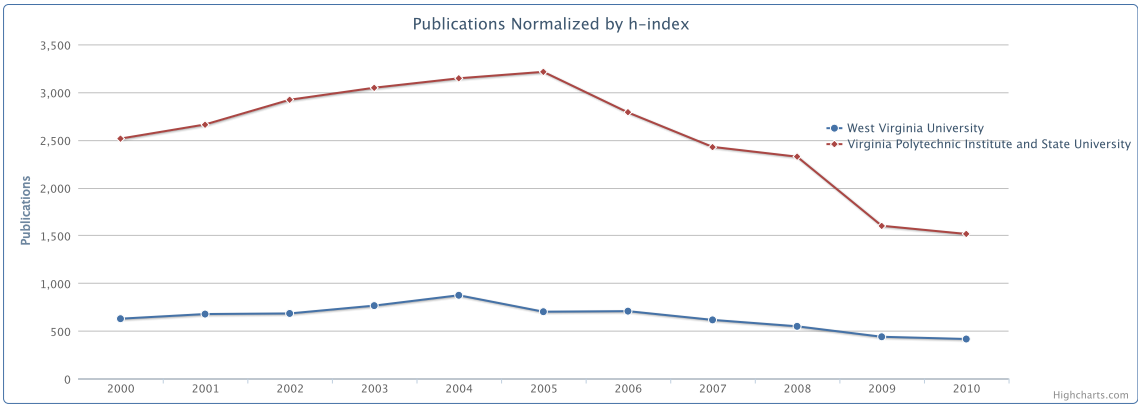

Figure 3.13: Publications normalized by h-index

Publication Counts are multiplied by the h-index value for each year. The universities' y-axis value will grow proportionately with the h-index. A higher quality (higher h-index) set of papers for the year is another way of measuring research productivity.

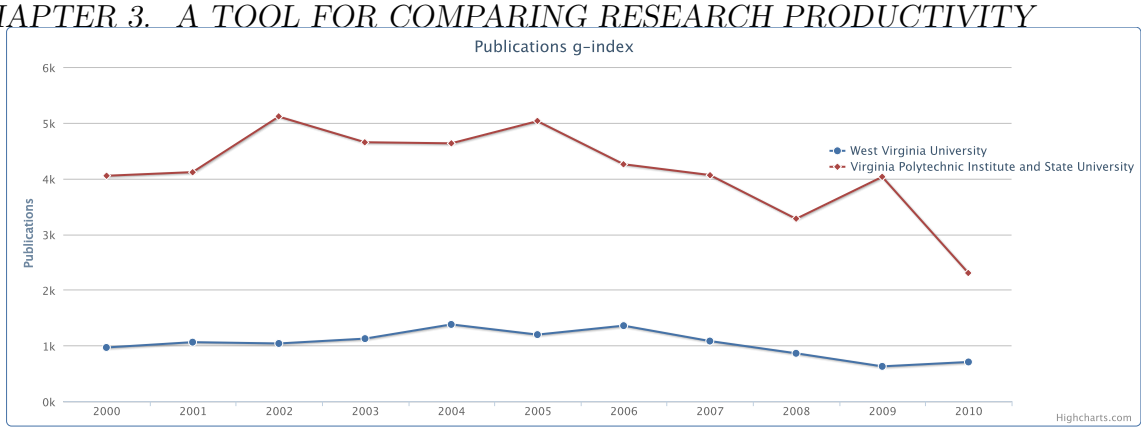

Figure 3.14: Publications normalized by g-index

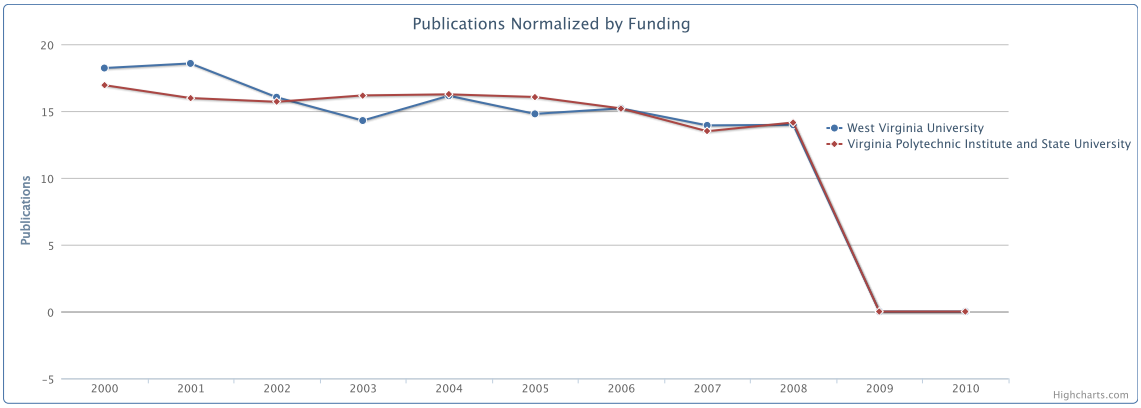

Figure 3.15: Publications per Millions of Dollars

After 2008, there is no funding data, so the graph goes to zero. WVU is actually close to Virginia Tech if you compare Publications per Millions of Dollars.

Chapter 4

Conclusion

Three different tools were created utilizing PHP, XHTML, JavaScript, and data from Microsoft Academic Search. The University Comparison Tool pulls citation and publication data then creates various calculated graphs. The Domain Popularity Analysis and Prediction tool allows for easy comparison of publication counts between two subdomains of a field. The Sanitization tool uses MAS data to direct web-crawling to PDF's and then searches those documents for keywords, disqualifying publications that possibly do not belong to authors.

There are many other analyses that could be performed on the available data. The tools that can be created from the Microsoft Academic Search data are only limited by the imagination and the API, which is in its infancy. It could provide easier access to some data and Microsoft perform some calculations on their own and provide those through a query. A more SQL-like system (without compromising security) for finer-grained queries would be more useful than the bulky queries they currently have.

4.1 Future Work

It would be interesting to compare the results we have gathered from the MAS database and compare the same results with data found in other databases such as Google Scholar. We could then compute some sort of findings, showing which database has more publications in certain domains.

Interfaces can always be improved up to a point. The comparison tool was designed with a minimalistic approach. No frills were added so one will not be distracted from

the results. A CSS theme could be adopted into the site giving it more personality and providing a possible better look and feel.

One the disadvantages of AJAX is that by default browsers do not cache AJAX data as well as standard web pages. This means that if a user pulls down data comparing two universities then requests different data after, the previous data will be lost to the browser. There are various ways to cache AJAX data in browsers. The easiest way would be to store each JSON response from the PHP code in a JavaScript array. Then, using back and forward buttons created by HTML (not the browser buttons) we could move back and forward through previous requests. We would just re-graph the data after each button press.

While the application is multi-threaded and usable by multiple users at the same time, it is limited by maximum requests per minute. Currently, if two users make large queries at the same time the API will shut the application user ID out for a short period of time. A solution to this would be to request more API keys or request a larger maximum requests per minute.

If a user requested WVU data from years 1990 to 2010, but then later requested data from 1980 to 2010, we are currently requesting the 1990-2010 data again. We could cache data in the JavaScript and then access it later for future requests. That way, we would only have to request data from 1980-1990, reducing processing time.

Currently API and network errors fail silently. The user will see that something is wrong, usually by data on a graph not forming correctly. Erroneous data is usually null. We could check for null data and submit new requests to the API until correct data is received.

References

[1] Ajiferuke, I., Wolfram, D. (2010). Citer analysis as a measure of research impact: Library and information science as a case study. *Scientometrics*, vol. 83, pp. 267-288, Issue 3.

[2] The Center for Measuring University Performance. `http://mup.asu.edu/research_data.html`

[3] Costas R., Bordons M. (2008). Is g-index better than h-index? An exploratory study at the individual level. *Scientometrics*, vol. 77, pp. 267–288, Issue 2.

[4] Egghe L. (2006). Theory and practise of the g-index. *Scientometrics*, vol. 69, pp. 131–152, Issue 1.

[5] Hirsch, J. E. (2005). An index to quantify an individuals scientific research output. *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 46, pp. 16569–16572.

[6] Hirsch, J. E. (2010). An index to quantify an individual's scientific research output that takes into account the effect of multiple coauthorship. *Scientometrics*, vol. 85, pp. 741–754, issue 3.

[7] Microsoft Academic Search API. (2012). `http://academic.research.microsoft.com/about/Microsoft%20Academic%20Search%20API%20User%20Manual.pdf`

[8] Moon T., Stirling W. (2000) Mathmatical Methods and Algorithms for Signal Processing. *Section: Linear Regression* pp. 147–149.

[9] Nie, Z., Ma, Y., Shi S., Wen J.,Ma W. (2007). Web Object Retrieval. *World Wide Web Conference Series* pp. 81–90.

[10] Sidiropoulos, A., Katsaros, D., Manolopoulos, Y. (2007). Generalized Hirsch h-index for disclosing latent facts in citation networks. *Scientometrics*, vol. 72, pp. 253–280, Issue 2.